



UNORYN

UNORYN Whitepaper v0.1

Protocol Design Draft | English edition | Review 3 | September 12, 2026

Genesis / Prototype Stage

Proposed Design

This paper is for engineers considering implementation, researchers examining the mechanisms and partners reviewing the project. It specifies how a controlled task network would receive authorized inputs, distribute bounded work, verify results and record contributions, and what evidence would justify further development. It is not a deployed protocol specification or security audit and creates no token or economic entitlement.

The operating-status baseline is the September 11, 2026 handover. By September 12, bilingual static website revision files and documents existed for local preview. That establishes preparation of presentation materials, not an updated live deployment. Acceptance evidence is still missing for a Node implementation, a multi-device demonstration, public enrollment, a Dashboard backed by real events, external repeat use and revenue. All stacks, fields, interfaces, thresholds and operational policies below are proposed. “Must” states a design requirement, not an implemented capability.

Contents

1	Abstract and reading conventions
2	Problem and related work
3	Initial workload and demand hypothesis
4	Architecture and trust
5	Node identity and registration
6	Task lifecycle and acceptance
7	Runner and reproducible environments
8	Verification policy
9	Contribution Graph data and provenance
10	Verified Delta evaluation design
11	Contribution accounting and reputation
12	Economic coordination and funding constraints
13	Threats, security and data rights
14	Governance, upgrades, disputes and recovery
15	Preconditions for a more open network
16	Experiments, gates and failure conditions
17	Open questions and reference use
	Appendix A Minimum entity model
	Appendix B Proposed interfaces
	Appendix C Idempotency and transaction pseudocode
	Appendix D State transitions and metrics
	Appendix E Minimum acceptance matrix
	Appendix F Reference index
	Source links

1 Abstract and reading conventions

UNORYN proposes to serve small development teams that need participants to reproduce checks. Genesis starts with offline contract-text rules, testing the loop from Node registration and compatible-task claiming through local execution, submission, verification, contribution records and display. A later authorized small-model local evaluation would test whether this evidence workflow supports a real AI workload.

The initial system depends on a central Coordinator and database. Nodes may disconnect, duplicate submissions or fail; networks may delay or lose messages. Scheduling assumes at-least-once delivery. Lease generations, submission idempotency and unique acceptance slots constrain duplicate accounting. They do not prevent every repeated computation or promise whole-network exactly-once execution.

Contribution Graph records participants, attempts, artifacts and reviews. Verified Delta records candidate-baseline comparisons under frozen evaluation protocols. Acceptance does not require improvement, and a record does not prove originality or reward rights. The mechanisms use ordinary provenance, experiment tracking and transactions; added value must be demonstrated through cross-participant reproduction efficiency and demand experiments.

Begin with a controlled prototype without a native token, arbitrary tasks or asset custody. Compensation, pricing, licenses, governance entities and any token need separate decisions. Continuation depends on real second use, tolerable verification overhead, recoverable service and supportable security obligations.

1.1 Terms and states

Contributor is a participant record, not necessarily a verified real-world person. Node is an authorized client-instance identity, not an independent computer. Task is a logical work request. Attempt is one execution attempt for a slot; Assignment is its lease allocation. Artifact is a versioned output or input object. Result is an envelope bound to one Attempt. Verification is a policy-based finding about a Result. Contribution Record records accepted work.

Completed is a Node's claim that execution ended and is stored only as telemetry. Submitted means the service received a structurally valid result. Verified means the required technical checks passed. Accepted means formal acceptance after policy, eligibility and transaction constraints. Failed means a logical task exhausted its allowed attempts or reached an unrecoverable outcome. A Node process error alone is an Attempt failure, not necessarily a Task failure.

“Known” refers to supplied evidence; “proposed” identifies design choices; “illustrative/synthetic” identifies teaching or scenario numbers. Project achievements require their own run records. External project documentation cannot establish UNORYN implementation. References use R identifiers and dated source checks; a provider's mechanism description is not independent performance measurement.

1.2 Document map

Sections 2-3 cover demand and differentiation; 4-8 cover controlled execution and verification; 9-11 cover provenance, evaluation and accounting; 12-15 cover economics, security and control; 16 defines experimental gates. Appendices A-E contain entities, interfaces, transactions, states and acceptance tests. Implement tests from the appendices before coding only the success path.

2 Problem and related work

2.1 Coordinating evidence as well as execution

A maintainer who sends a script to a colleague still needs to know whether inputs matched, dependencies changed, output was edited and a conclusion relies on an obsolete version. Simple tasks already have execution tools. Cross-boundary collaboration needs consistent evidence handover. UNORYN would connect requests, attempts, verification and revisions so consumers can explain acceptance.

Not every user needs this. A one-off personal task usually needs no network; a team with CI and experiment management may only need extra export fields. Existing tools can already record contributions in many forms. A competitor's lack of an identically named feature is not differentiation. Current evidence supports neither “world first” nor technical-leadership claims.

2.2 Comparison method

Compare coordination objects, users, verification, trust, attribution and economics. External capabilities below reflect official documentation, not measured deployment performance. Every UNORYN comparison is a hypothesis, not an existing advantage.

Bitcoin coordinates transaction history and double spending for payment participants, using signatures, a proof-of-work chain and an honest-hashpower assumption. Fees and issuance support incentives; attribution concerns authorization and block production, not AI authorship. [R1] The relevant lesson is explicit validity and trust assumptions. Arbitrary AI outputs do not share a simple global validity predicate; replacing hash puzzles with useful computation does not inherit Bitcoin's security.

GEOD is GEODNET's token. Its documentation links RTK data services, base-station participation, availability and measurement-quality criteria with rewards. [R2-R3] Users include positioning-service consumers and station operators. Attribution concerns station service under device, measurement and platform rules. Service revenue and independent quality have not been verified here. UNORYN lacks similarly demonstrated paying demand and cannot substitute copied token allocations for evidence.

Gensyn documents execution, verification, communication and coordination for machine intelligence, including incentives and payments. [R4] REE describes reproducible inference on supported hardware, with ordinary, same-hardware deterministic and reproducible modes; the reviewed page identifies v0.6.0 and proprietary components. [R5] These are substantial overlaps with UNORYN. Verification depends on actual implementation coverage and does not establish method authorship. This paper measures neither performance nor economics and does not reduce Gensyn to compute rental. Any UNORYN distinction must be narrower, such as evidence usability.

Bittensor subnets coordinate service provision and utility assessment. Official Yuma documentation describes validator scores and stake-weighted coordination and incentives, with honest-majority-stake assumptions in its resistance arguments. [R6] Attribution is based on evaluated service utility, not universal proof of originality. UNORYN has no permissionless validator system and cannot claim superiority simply by naming several contribution types.

io.net documents compute supply, periodic worker proof-of-work challenges, memory checks and payment options including IO, USDC and cards. [R13-R13b] Its users are compute consumers and providers; attribution

concerns device and task service under platform checks. These documents do not establish arbitrary model quality, and this review makes no independent performance claim. UNORYN still needs evidence that cross-artifact review creates distinct demand.

W3C PROV supplies entity, activity and agent concepts; OpenLineage describes jobs, runs, datasets and events; MLflow tracks parameters, code versions, metrics and artifacts. [R7-R9] These systems rely on their record producers and access controls, and do not automatically prove real execution or create reward rights. UNORYN should reuse their concepts and add exportable delegation, verification policy and acceptance decisions where useful. Graph relationships themselves are not new technology.

Celery documents familiar retry and idempotency practices. [R10] Scheduling is ordinary engineering, not a defensible innovation by itself. If existing tools plus a thin evidence-export layer suffice, prefer that combination and later adapters over mandatory migration.

2.3 Differentiation hypothesis

Test whether external collaborators using one evidence package answer “Why was this accepted?”, “How can I reproduce it?” and “What is no longer valid?” faster. Compare local scripts plus structured logs or MLflow, measuring tracing time, missing fields and successful error localization. If no improvement appears, reduce the graph to a standard export rather than invest in a graph database or elaborate interface.

Network effects are unproven. More nodes may widen environment coverage but increase review and support costs. More contribution categories may enrich records but complicate permissions and attribution. Wider expansion is justified only when different participants repeatedly reuse evidence and the additional coordination cost remains manageable.

3 Initial workload and demand hypothesis

3.1 Three candidates

Task	Inputs and consumers	Main acceptance method	Burden for two people
Offline contract-text rules	Authorized samples; tool maintainers	Known answers and reference implementation	Lowest; suitable for protocol tests
Local small-model classification	Authorized model and labels; evaluators	Row-level output and frozen metrics	Moderate; device differences
Real-time crypto intelligence	Changing sources; information consumers	Provenance, freshness and subjective rubric	High; weak repeatability

Use the first candidate only as a task-network loop test. It executes no contract, signs no transaction and touches no assets. Output describes text patterns or parsing status, not audit conclusions or AI capability. Preserve this scope in publicity and the Dashboard task_family label.

3.2 Task package

Initially, the team writes a small set of publishable example texts with explicit permission and expected patterns. Third-party samples require source, version, license and redistribution/derivative-use restrictions. Unverified permissions prevent distribution to external Nodes. Internally licensed material must not become publicly accessible artifact content.

The input manifest includes `sample_id`, `relative_path`, `byte_length`, `content_digest`, `license_ref` and `expected_encoding`. Reject path traversal, symlink escapes, decompression bombs and duplicate-name ambiguity. Maintainers control rule packages. Do not fetch dependencies from sample URLs or execute scripts, macros, compiler plugins or dynamic imports embedded in data.

Freeze `result_schema_version`. JSON output includes rule versions, row-level decisions, match ranges, digests and unsupported reasons. Each (`sample_id`, `rule_id`) has exactly one row with `MATCH`, `NO_MATCH` or `UNSUPPORTED`. `MATCH` requires a nonempty matches array; `NO_MATCH` requires an empty array; `UNSUPPORTED` requires a reason and cannot become binary 0. Errors must not leak environment variables, private paths or credentials.

Locations use half-open intervals [`start_byte`, `end_byte`) in the original UTF-8 file: $0 \leq \text{start_byte} < \text{end_byte} \leq \text{byte_length}$, with both ends on character boundaries. Do not silently normalize CRLF, strip a BOM or normalize Unicode. Freeze BOM support by task family. Sort and deduplicate ranges by start then end; the rule defines whether overlap is allowed. Freeze case sensitivity, comment/string treatment, syntax coverage and multiple-match semantics so independent implementations compare the same specification.

Reject duplicate JSON keys, nonfinite numbers and invalid Unicode. Apply RFC 8785 to applicable JSON objects; hash raw bytes for samples and binary artifacts. [R12] The schema defines array order: sort result rows by `sample_id` then `rule_id` and compare exact IDs. Sorting object properties does not make arrays unordered. Encode digests as sha256: followed by 64 lowercase hexadecimal characters. Exclude an object's own digest field when calculating it.

3.3 Execution limits

Initially support one tested OS/architecture combination, CPU execution, read-only inputs, temporary outputs and no task-time network access. Illustrative engineering starting values, not measurements: 60 seconds, 512 MiB memory, 10 MiB input and 2 MiB output per attempt. Use them to test limit enforcement; measure supported machines before freezing real-task limits. Resource violations have explicit error codes and cannot create unlimited retries.

Parameters accept whitelisted rules and numeric options, not shell command strings. Test long lines, invalid UTF-8, nested structures and parser failures even with trusted samples. Internal limits are not a complete sandbox; public distribution needs Section 13's additional gates.

3.4 Users and payment

An open-source maintainer may request a rule-version comparison; a colleague or researcher may consume it; a team needing ongoing hosted review and support might pay. These roles may overlap, but Node supply is not demand and no order or payment evidence is supplied.

Use the same real task to compare the user's existing script process against UNORYN's evidence process, including total time and errors. Counterbalance trial order to reduce learning effects. Small samples do not establish statistical representativeness; investigate specific reasons for reuse. Unexecuted quotations are interest, not revenue.

A proposed G3 gate is at least two non-team users returning with authorized tasks without promised rewards and identifying a decision informed by the result. Initial interviews may involve 3-5 people. These are minimum observations, not market proof. Team-entered tasks, promotional engagement and airdrop expectations do not validate demand.

3.5 Later local AI workload

After the loop stabilizes, add a compact model whose license permits offline use to classify fixed text samples into binary or finite categories. Record model/tokenizer digests, backend, quantization, threads and randomness settings. Do not invent a model version, license clearance or performance claim.

Acceptance requires disconnected operation, process and resource logs, row-level predictions and reference-environment review. Controlled-device evidence can support a program-run claim but is not cryptographic proof on a malicious remote machine. Mark cloud APIs as separate providers in cost comparisons; never count them as local inference on a participating computer.

4 Architecture and trust

4.1 Three domains and six logical components

Execution domains are the submission/management side, coordinating service and contributor machine. Logical components are Task Registry, Coordinator, Node Runner, Verifier, Contribution Store and read-only Dashboard. They may share one service process or database; six separate deployments are unnecessary.

The proposed prototype uses FastAPI, a Python CLI Node and SQLite. Artifacts reside in controlled, content-addressed storage with authorization checks. This stack has not been deployed or load-tested. A database work table can poll verification jobs without introducing a message broker, graph database, P2P or blockchain simultaneously.

Data flows from a frozen package and task review through Node lease, authorized download, local execution, result submission, verification queue, acceptance transaction, query and evidence export. Storage manages bytes, the database manages relationships and state, and the verifier applies policy. None substitutes for the others.

4.2 Control boundaries

Maintainers approve participants, publish Runners, define task families, pause dispatch and revoke credentials. The Coordinator controls queues and acceptance. Verifiers may reject invalid outputs but must not modify inputs or reward rules. The Dashboard cannot write acceptance state, and requesters cannot directly certify their own results.

A two-person team may lack institutional independence. Separate privileges and review steps reduce mistakes but shared control must be disclosed. Use individual credentials; one shared administrator account prevents reliable attribution. Two-person approval adds procedural checks, not external independence.

4.3 Initial assumptions

Assume the Coordinator follows published policy, operating systems are basically trusted and controlled participants follow test rules. Networks may fail and Nodes may crash. Server time governs leases and deadlines; client time is diagnostic. Storage needs corruption, loss and recovery tests. A local transaction is not high availability or administrator-proof history.

Genesis has no consensus protection against a malicious Coordinator, which could censor access, delay work or rewrite history not retained externally. Exportable records and audit digests can reduce concealed-change risk, but participants still trust the operator. Reassess this before permissionless or paid operation.

4.4 Separate evolution dimensions

More execution devices affect capacity and environment coverage; independent verifiers affect review trust; distributed governance affects unilateral control. Multiple data centers under one operator remain centralized. Anchored hashes do not decentralize control of underlying data or Runners.

5 Node identity and registration

The service generates opaque Contributor IDs linked to participation-terms versions and permissions. Each Node binds contributor_id, client version, device claims and a revocable credential. device_ref is a controlled inventory reference, not justification for permanent tracking through MAC addresses or fingerprints. Public views use pseudonyms and coarse compatibility information.

Capabilities include OS, architecture, CPU/RAM, accelerators, supported Runner versions, limits and available slots. Treat claims as scheduling hints followed by small compatibility tests. Passing does not prove enduring, exclusive hardware ownership or absence of outsourcing. Claimed hardware and time do not directly create high points.

5.1 Credentials

Maintainers issue single-use invitations for controlled tests. Store secure credential-verification material, not reusable secrets. A Node may claim its own leases, submit its own results and read its own status, not create arbitrary tasks or access other participants' artifacts. Public communication requires properly configured TLS. A loopback-only development mode must not reach public deployment.

Rotation creates a new key_id while preserving revocation timestamps. Revocation immediately prevents new claims and submissions. Keep previously accepted history; if compromise affects earlier evidence, mark the affected window and reverify. Deletion cannot erase an incident. Later signatures still need theft, rotation and binding controls.

5.2 Heartbeat and exit

Heartbeats report availability, the current attempt_id and local resources, not a guarantee of completion. Renewal is a separate operation tied to the active Attempt and generation and bounded by the task deadline and maximum execution window. Fixed finite leases are recommended initially to avoid renewal complexity.

Normal exit stops claiming and may finish a valid lease or abandon it. Server scans handle unreported disconnection. Revocation, pause and maintenance are not evidence of maliciousness; penalties require specific wrongdoing rather than unreliable connectivity.

5.3 Independence records

Record physical machines, operators, network conditions and implementation origins separately. Participant declarations supported by video or in-person checks may establish controlled inventory, with minimal sensitive collection and explicit trust limits. Two machines owned by one person support hardware-difference testing, not independent-operator claims. Proxies can create different IPs; shared offices can use one IP. Neither proves or disproves a Sybil attack.

6 Task lifecycle and acceptance

6.1 Logical tasks and slots

Task creation freezes `spec_digest`, inputs, Runner, verification policy, permissions, deadlines and maximum attempts. Changes require a new task version. Every Task has `task_role=primary` or `validation` and one primary acceptance slot; the slot's name does not define the task's purpose. Validation tasks require `verifies_task_id` pointing to a primary Task. Do not recursively generate validation contributions. Each slot can have at most one historical Acceptance; revocation changes effectiveness, not that count.

Also freeze `acceptance_policy_id/digest` and `accounting_policy_id/digest`. Verification policy governs technical checks, acceptance policy governs admission, and accounting policy governs units and counts. Do not read an undefined global current policy at execution time. Current permissions and revocation are additional eligibility checks, not changes to frozen criteria. TaskSpec's digest covers fixed semantics and excludes state, slots, generation time and `spec_digest` itself; `spec_version` defines the field projection and canonicalization version.

Each slot has an integer generation. A new Attempt increments it and creates a finite expiry and unguessable lease token. A Result binds `task_id`, `slot_id`, `attempt_id`, generation and package digests. New submissions carry the secret lease token in a separate authentication field, checked against the Attempt. Do not include it in Results, content-deduplication digests or public logs. The stable actor identity is Node ID, so rotating `key_id` does not reset the idempotency namespace. An old generation's correct answer cannot override the active generation.

6.2 Normal path

A Task moves `CREATED` to `QUEUED` after permission and package checks. A compatible claim selects a slot, inserts an Attempt and sets `IN_PROGRESS` in a short transaction. No available work returns an explicit empty response rather than a fabricated task. The Node checks the package, reports start and submits a result envelope after execution.

One short submission transaction rechecks Node permissions, bindings, active generation, server deadlines, cancellation, structure and digests, then inserts immutable Result and verification work. The Attempt becomes `SUBMITTED`. A timely committed submission transfers responsibility to server-side verification; expiry of the old execution lease must not reassign the main task.

The verifier checks frozen content outside a transaction. PASS means technical checks passed. The acceptance transaction rechecks IN_PROGRESS, review deadline, acceptance switches, permissions, nonrevoked versions, frozen verification requirements and an empty acceptance slot. It atomically writes Acceptance and ContributionRecord and sets ACCEPTED. Duplicate verification completions still permit only one acceptance.

6.3 Late results, retries and cancellation

Set received_at once from server time inside the write transaction after bytes are durably stored and eligibility is checked. It must be strictly before lease expiry and execution_deadline. That time is effective only if the transaction commits; rollback creates no valid submission. The success response may arrive after the deadline without changing eligibility. Network arrival and client-signed time are not eligibility timestamps. Database waiting can make a request late and should be explained to clients; lateness alone is not cheating.

Late results may leave minimal rejection audit events but enter neither valid Result queues nor contribution accounting. Failed or expired execution returns the slot to QUEUED only while attempt budget and the task deadline allow. New Attempts use a new generation and backoff. Exhaustion becomes FAILED with a reason: unsupported input, Runner defect, incompatibility, resource breach or temporary infrastructure failure. Do not retry every category indefinitely.

Cancellation sets an unaccepted Task to CANCELLED and removes active-slot eligibility in a transaction. Database commit order resolves races: cancellation first prevents later submission/acceptance; acceptance first makes cancellation conflict and requires a separate revocation. Emergency stop should terminate local work promptly, but server cancellation cannot guarantee an immediate remote-process stop.

6.4 Verification failure and restart

FAIL means the result is invalid and preserves evidence; policy determines whether to rerun execution. ERROR means verifier crash, timeout or unavailable infrastructure and does not establish Node fault or justify point deductions. Verification has its own recoverable leases. Recovery retries the same Result rather than forcing Node execution again.

Limit verification retries and set review_deadline. Repeated ERROR makes the Task BLOCKED, explains infrastructure unavailability and stops automatic reassignment. Acceptance's server eligibility time must be strictly before review_deadline. Deadline expiry or exhausted ERROR retries blocks both acceptance and redispach. Maintainers may restore ERROR-blocked verification to IN_PROGRESS with a reason only before the original review deadline. Once expired, do not extend the original task: retain diagnostics and create a replacement Task if needed. Time continues during an acceptance pause.

Restart reconstructs queues, active leases, expiry work and verification jobs from the database. Memory queues are only optimizations. SUBMITTED must not be mistaken for running execution and reassigned. An external broker would need a transactional outbox and duplicate-delivery handling; a database work table is simpler initially.

6.5 Idempotency and atomicity

Clients reuse keys for retries. Scope keys by at least actor_id, route and key, with the normalized request digest and original response. Same key/content returns that response; changed content conflicts. A success boolean is insufficient because the client needs the original Attempt or receipt. Authenticate current permissions before replaying responses, including for revoked credentials.

Unique constraints permit at most one valid Result per Attempt, one historical Acceptance per main slot and one initial ContributionRecord per Acceptance. A content hash alone is not a contribution uniqueness key: separately commissioned checks can legitimately yield identical bytes. Deduplicating by contributor_id would incorrectly discard that person's different tasks.

SQLite allows multiple readers but one writer at a time. [R11] Use short BEGIN IMMEDIATE transactions; enable and verify PRAGMA foreign_keys=ON on every connection outside a transaction; limit busy retries; perform downloads and verification outside the write transaction. [R14] Busy, disk-full and commit errors must not receive false success responses. Transactions constrain this database, not storage durability, backups, external payment or malicious administrators.

7 Runner and reproducible environments

A Runner is a reviewed program for a task family, not an arbitrary command agent. Its release manifest records runner_id, version, source reference, dependency lock, input/output schemas, supported environments, limits and package digest. Versions aid readers; digests identify actual bytes. Retain both to detect repackaging under the same version.

Minimize dependencies and separate installation from execution. Tasks install nothing, read no arbitrary configuration and call no external model API. Environment manifests include Python, important libraries, OS, architecture and encoding; AI workloads add model, tokenizer, quantization, backend and driver details. Downloadability is not license permission or package-authenticity review.

Distinguish identical normalized discrete outputs, numerical agreement within frozen tolerances, and statistical/metric compatibility under an agreed plan. Each Task chooses its supported class. Legitimate floating-point differences are not automatically cheating; broad tolerances must not conceal material regression. Unsupported environments return unsupported until tested.

Limits cover duration, memory, disk, output size, CPU/accelerator use and network access. Test actual OS enforcement separately from application declarations. An unenforced platform is not publicly supported. User stop takes precedence over points and leases.

Create outputs only in isolated work directories. Clean only Runner-created temporary files under a retention policy, never recursively delete arbitrary user paths. Sanitize logs and keep environment details local by default. Defer automatic updates initially; future updates still require authenticity verification, revocation and rollback.

8 Verification policy

8.1 Layers

Check permissions, schema, size, digests and bindings first; task semantics second; reproduction or sampling under policy third; acceptance eligibility fourth. Passing one layer does not imply passing later layers. Record outcomes and reason codes.

At creation, freeze policy_id, version, reference implementation, test data, review requirements, tolerance and environments. Changed references require new policy and Verification records and possible revocations, not overwriting historical PASS. Verifiers must not treat a submitted result as their own answer key.

8.2 Deterministic workloads

Use known answers to test the Runner, then independently written reference rules or manually reviewed golden outputs to check tasks. Rerunning identical code on another machine checks environment and transport consistency. Independent implementations may find shared code defects but can share a mistaken specification; retain counterexamples, boundary tests and human review.

Predictable tasks are easy to copy. Binding, hidden labels and delayed answer publication are proposed controls with unmeasured detection, false-positive and cost characteristics. Once sent to a Node, an input is no longer hidden from it and can be forwarded. Noncash points can still motivate farming, reputation competition or airdrop expectations. Actively test correct-answer copying, outsourcing and shared errors; distinguish blocked, observed and undetectable behavior. Do not claim proof of useful work before evidence exists.

8.3 Model and subjective workloads

Store predictions, label versions and all runs. Separate parse coverage, quality and resources. Missing outputs and timeouts count in coverage and budgets instead of disappearing from scores. Freeze repeat counts and sampling units and report hardware differences. Use declared tolerances and statistical plans where bitwise identity is unavailable.

Subjective work requires a rubric, disclosed reviewer roles, blind-review arrangements and disagreement handling. Shared bias survives unanimous votes. Genesis does not automatically turn subjective opinion into rewards. Retain a human service or reject the workload if reliability and disputes make automation inappropriate.

8.4 Sampling limits

Consider sampling only when full verification is too expensive and risk permits it. Policy, not the submitter, controls sample selection. No detected error does not prove all outputs correct. Independent-error models can yield detection probabilities, but collusion, clustered errors and adaptive attacks violate those assumptions. No universal safe sampling rate is supplied.

Measure hidden-set maintenance, human disputes, reruns and idle verification capacity. Comparing only execution expenses cannot establish lower platform costs. Section 12 defines the framework and Section 16 requires actual measurements.

9 Contribution Graph data and provenance

9.1 Entities are not value proofs

The graph is a relationship and evidence index that can use relational storage. Contributor identifies who is registered; Node identifies client credentials; Attempt identifies authorized work; Artifact identifies bytes; Verification identifies a policy-based review; ContributionRecord identifies accepted work. A wallet-address hash object cannot stand in for all of them.

Task has an opaque ID plus spec_version and spec_digest. Result binds attempt_id, input_digest, output_artifact_id and environment_digest. Verification binds result_id, policy_digest, verifier_version and evidence_artifact_id. Validate existence and permission for every reference; dangling objects cannot appear as verified work.

The service generates UTC created_at, received_at and accepted_at. Client timestamps use claimed_ prefixes and are not trusted billing clocks. schema_version describes structure while record_version describes revision. Specify the digest algorithm and canonicalization version.

9.2 Edges and the DAG

derived_from links particular artifact versions and must reject self-links and cycles. uses records task inputs; produced_by associates an artifact with an Attempt without proving production occurred; verified_by connects Result to Verification; credited_to connects a contribution to a participant. disputes, revokes and supersedes are review references and need not obey derivation-DAG rules.

Retain every actual data/code dependency instead of merging provenance for a prettier graph. The visualization is a query projection; stable IDs and relationships are the evidence and must be exportable.

9.3 Duplication, attribution and reuse

Identical bytes may share a blob while Artifact entities differ in license, claims or version meaning. Repeated submission to the same task slot adds no contribution. Separately commissioned reviews can count as Validation despite identical output, but require a stated purpose. Uncommissioned repetitions do not automatically become paid demand.

Submitters do not automatically become method authors. Authorship, derivation and execution claims need a claimant, evidence, reviewer and status; unreviewed claims remain unverified. Licenses support use scope, not authorship by themselves. source_ref, author_claims and license records support review; earliest upload or a blockchain address cannot alone establish originality. Adoption records neither create revenue shares nor establish commercial-use permission.

Economic rights require separately agreed task terms, permissions and settlement policy. Genesis creates no perpetual upstream dividend. Record joint authors and disputes for combined methods. If allocation is unreliable, defer it rather than invent exact intelligence-contribution percentages.

9.4 Revisions, revocations and audit

Do not edit accepted objects' key fields in place. Corrections create versions with reasons, actor, old reference and new verification. Revocation adds events and necessary adjustments without deleting historical acceptance. Current-effective views exclude revoked records; history retains them.

Policy reassessment must identify affected task families and Runner, data or verifier versions, pause relevant work and reverify. Never migrate old PASS into a new policy automatically. Recompute points after revocation and retain before/after differences.

Append-only logs record actor, action, object, before/after digests, UTC time and reason. A hash chain helps detect rewriting only with a trusted retained reference such as an externally stored old head; a local administrator can otherwise rewrite the entire chain. Begin with exports to another controlled location and later evaluate signing or public anchoring.

9.5 What evidence can establish

Database records establish policy-based recorded events to the extent permissions, logs and backups are trusted. Hashes support byte consistency, signatures key-message authentication. Neither automatically proves execution, independence, authorship, rights or semantic correctness. Anchoring does not repair an invalid license or answer key.

An export includes frozen Task, Result, Artifact manifest, Verification, Acceptance, current revocation snapshot and environment manifest, plus exported_at and access scope. Offline checks describe the snapshot time, not continuing validity after export; authorized current-state retrieval is needed for later revocation. Exclude secrets and private paths. Reviewers can check digests and references offline, but restricted inputs mean restricted reproducibility, not universal public reproduction.

10 Verified Delta evaluation design

10.1 Definition

Verified Delta is a record and review procedure for comparing a candidate against a baseline within an agreed scope. Its foundation is comparative evaluation. It neither defines universal intelligence units nor makes every contribution additive, and it cannot replace demand validation. Ordinary tasks need only satisfy their own acceptance rules. Comparisons explicitly declare baseline and candidate.

Protocol P includes baseline B, candidate C, data D, environment E, budget U, randomness plan S, metrics M, minimum practically relevant gain and resource guardrails. It also records candidate-selection history, holdout-access rules and the review plan. Bind B/C/D/E/U/S/M to versions and digests. Any change requires a new protocol_id or revision rather than combining incompatible experiments.

10.2 Data and randomness

Development data selects candidates; a holdout evaluates them under a frozen plan. Public examples can test the workflow but do not establish generalization if candidate authors saw answers. Begin with public-set process tests and later use new, authorized holdouts for suitable formal comparisons, labeling the distinction.

Independence may lie at the source-project, device, seed or batch level. Running one sample a hundred times does not produce a hundred independent samples. Group related-project samples when estimating uncertainty and prevent near duplicates crossing training and test data. Record versions, sampling conditions and exclusions.

Random workloads need preregistered repeat counts or sequential stopping rules and retention of every run, including failures. Common seeds may reduce some variance but cannot ensure identical outputs across

backends. Confidence intervals must suit both the metric and dependence structure. Small samples or absent row-level outputs require an unreliable-estimate label, not decorative significance claims.

10.3 Quality, latency and cost

Let q_B and q_C be quality scores under the same protocol. Absolute gain $d_q = q_C - q_B$ is higher-is-better, in score units. Where q_B is nonzero and a ratio is meaningful, $r_q = d_q / q_B$ and percentage change is $100 * r_q$. For F1 on a 0-1 scale, percentage-point change is $100 * d_q$; it differs from relative percent.

Let l_B and l_C be latency on the same hardware with the same cold/warm-start definition, in ms/sample; savings $d_l = l_B - l_C$. Let c_B and c_C be unit costs under the same accounting boundary, in currency/sample; savings $d_c = c_B - c_C$. Different device prices can make latency and cost move in opposite directions. Report both.

Illustrative policy, not approved parameters: a quality-first task might require $d_q \geq 0.02$, candidate latency ≤ 1.10 times baseline and candidate cost ≤ 1.05 times baseline. Real thresholds follow task error costs and budgets, not observed results. If a metric lacks a meaningful ratio scale or its baseline is zero or negative, report absolute changes and units. Freeze the primary metric and nonregression guardrails before running; improved F1 cannot compensate for undisclosed time or cost violations.

10.4 Four outcomes

Verified Delta is the name of a proposed process, not proof that the process or project has been validated. Initial automatic decisions support only Section 10.6's fixed finite sets and deterministic binary outputs. Improvement means the frozen quality threshold and resource constraints were met on that set. It does not infer future samples, actual operators or commercial value.

Evidence eligibility comes first. Ineligible evidence yields INSUFFICIENT while preserving observed failures. With complete evidence, excessive quality loss or any resource breach yields DEGRADED with reasons. Otherwise gain $\geq \text{min_gain}$ yields IMPROVED, and smaller gain yields NOT_CONFIRMED. Every resource guardrail must pass.

Missing inputs, permissions, complete output, trusted measurement or review cannot yield the other outcomes. An observed overrun with incomplete quality evidence remains INSUFFICIENT with the overrun disclosed. Random workloads remain observational/insufficient until an appropriate statistical method is implemented; a default statistics function cannot promote them automatically.

10.5 Numerical illustration

All figures here are illustrative/synthetic, not UNORYN measurements. A set of 200 binary examples contains 100 positive and 100 negative labels. Baseline TP=80, FP=20, FN=20, TN=80. Candidate TP=86, FP=18, FN=14, TN=82. $F1 = 2TP / (2TP + FP + FN)$, so baseline is $160/200 = 0.8000$ and candidate is $172/204$, approximately 0.8431.

Still synthetic: absolute gain is approximately 0.043137, or 4.3137 percentage points, and relative gain approximately 5.3922%. Calling this a 5.39% accuracy improvement would be wrong: the metric is F1, and percentage points differ from relative percent. Consistent matrix totals support arithmetic, not proof of runs, representativeness or independence.

Still synthetic: latency rising from 100 to 112 ms/sample gives $d_l = -12$ ms and breaches an illustrative 1.10 ratio cap despite higher quality. With only aggregate matrices and no paired outputs, frozen scope or review, the conclusion remains INSUFFICIENT with a latency-breach observation. Complete measurement evidence is needed before confirming resource regression under the protocol.

No confidence interval is supplied. Marginal matrices cannot establish sample-level paired differences or justify a fabricated paired test. Retain both predictions for every sample and an appropriate preregistered comparison, failure report and uncertainty method.

10.6 Implementable finite-set decision procedure

Use a frozen manifest of (sample_id, rule_id) rows, each with a reviewed binary expected label. Positive means a specified text pattern, not a security flaw. The manifest is nonempty with at least one positive and one negative label. Both versions return exactly one 0/1 output per row. Missing, duplicate, extra or UNSUPPORTED rows, parse failures and timeouts make evidence insufficient. Retain failure counts and spent budget; do not remove failed rows to simulate completeness.

Freeze protocol_id, revision, protocol_digest and B/C/D/E/U/M digests before running. M uses $F1 = 2TP/(2TP+FP+FN)$. Required thresholds are $\min_gain > 0$, $\max_quality_loss \geq 0$, $\text{latency_ratio_cap} > 0$, $\text{cost_ratio_cap} > 0$ and the total budget. Task acceptance needs determine values; missing values prevent the comparison. Section 10.3's illustration is not a default.

Store thresholds as decimal strings and compare exact rational values. Compute F1 from integer-count fractions instead of rounded floating-point scores. Latency and cost reject NaN, Infinity, negatives and unitless values. Ratio baselines are strictly positive. Freeze pricing and allocation versions beforehand.

S is nonrandom in this initial scope. In one controlled reference environment, complete runs in B, C, C, B order. Both runs of each version must agree row by row. Repeats check determinism, not independent samples or confidence intervals. Do not choose the fastest/best run. Retain failed attempts and costs; a retry experiment receives a complete new revision.

For each run, latency is trusted-observer time from loading frozen input until the complete output is written, divided by row count, in ms/row. Freeze inclusion of loading, warm-up and caching. Pair the first B with the first C and the second B with the second C. Each pair requires $l_C/l_B \leq \text{latency_ratio_cap}$. These are observations, not stable-performance guarantees. Zero/negative baseline latency or invalid measurement makes evidence insufficient.

Likewise, each pair requires $c_C/c_B \leq \text{cost_ratio_cap}$. c is end-to-end evaluation cost per row in one currency, including execution, transfer, storage, verification and allocated support. Choose cash or economic cost in the protocol; do not mix them. Trusted usage with a frozen price list or verifiable bill is needed. Self-report, zero baseline cost, missing verification expense or execution-time-only measurement cannot support the ratio.

Give protocol-wide review, failures and shared support unique expense rows. Allocate them to runs by a pre-frozen rule; do not count a row again as an extra total expense. If allocation cannot be justified, retain the total and mark the ratio insufficient, not zero. Also report whether the complete experiment, including failures, exceeded its budget.

A controlled reviewer checks version digests, labels and outputs, all four runs, measurement provenance and cost allocation, and independently recomputes F1 and guardrails. Record permissions, control relationships and evidence in Verification. Review does not require a free extra full rerun; if needed, register count, expense and budget first. Human-reviewed representative counterexamples support labels, but trusted labels and observers still cannot exclude unknown defects.

Frozen P contains only pre-run versions, rules, budget, candidate history, access rules and review plan. protocol_digest excludes itself and future run/review results. A separate post-run evidence package references protocol_id, revision and protocol_digest and contains runs, measurements and reviews with its own digest. Do not rewrite P with outcomes.

Eligibility review checks P's freeze time, permissions and candidate history against that package. Public fixed-set comparisons use scope=public_finite_set, never an unseen-data or generalization claim. A claimed holdout requires additional access/leakage evidence; violations mean insufficient evidence. Submitter checkboxes cannot establish nonleakage or review.

After those eligibility checks, compute $d_q = q_C - q_B$. If $d_q < -\text{max_quality_loss}$, either paired resource ratio exceeds its cap, or total budget is exceeded, output DEGRADED. Otherwise $d_q \geq \text{min_gain}$ gives IMPROVED; otherwise NOT_CONFIRMED. Equality at a loss tolerance or resource cap passes the guardrail; equality at min_gain meets improvement. INSUFFICIENT takes precedence whenever qualification fails. Preserve reason_codes, values, units, scope and protocol digest.

These conditions can be coded and tested, but an end-to-end implementation is still missing. Random models, continuous errors, future-sample inference and subjective scoring require their own statistical plans and are not automatically supported. A numerical function returning IMPROVED without eligibility evidence is only a calculation example, not a public Verified Delta report.

10.7 Screening and joint attribution

Record failed and abandoned candidates. Repeated holdout access, best-seed selection, metric switching and reporting only wins create bias. Future-sample inference after multiple candidate comparisons needs a defined comparison family and suitable error control or independent final confirmation. Fixed public-set comparisons report only that set and retain all candidates. Leakage invalidates the evaluation protocol; preserve history and use new data.

Changes from methods A and B individually generally do not sum to their combined effect. Ablations offer local evidence dependent on baseline, order and dataset. Genesis does not estimate universal individual marginal intelligence value. Record joint work; separately agree economic allocation and pause allocation while attribution is disputed.

11 Contribution accounting and reputation

Accounting serves diagnostics and acceptance facts, not promotional points. Compute initially counts accepted primary-result submissions, not CPU/GPU volume or proven operators. Validation counts commissioned, accepted validation submissions supported by evidence. Knowledge and Intelligence record licensed artifacts and adoption. Different units remain separate, without a universal value leaderboard.

Genesis may use simple counts or task-family weights announced and versioned in advance. Weights are not market prices. Claimed GPU models, uptime or arbitrary uploads do not generate unlimited points. Uncommissioned duplicates, late/invalid submissions and slot replays add no effective contribution.

Reputation separately reports accepted proportion, timeliness, review coverage and revocation rate by family, with sample sizes and windows. Small samples neither discredit newcomers nor certify universal expertise. Disclose difficulty and scheduling-selection bias rather than rank raw pass rates as a universal rights system.

Points have no cash, conversion, airdrop or return entitlement. Corrections need reasons and evidence. Infrastructure errors do not deduct points. Suspected violations may restrict claims or open disputes while participants supply evidence. Unclear responsibility cannot trigger automatic economic punishment. No slashing, fee or custody mechanism is approved here.

Dashboard measures: submitted_tasks counts registered distinct task_id; execution_attempts counts attempt_id; submitted_results counts valid result_id; verified_results counts distinct current results meeting frozen policy. Split each by Task.task_role. accepted_tasks counts primary tasks with currently effective Acceptance; accepted_validation_tasks counts validation tasks separately; revoked_acceptances is separate. Distinguish team demonstrations from external requests. An accepted task is not automatically a paid order. Repeated verification adds no new verified result, and a validation task's primary slot does not place it in customer demand.

Completed remains telemetry. Define failure-rate denominators and periods, for example FAILED among primary logical tasks that reached a terminal state in the cohort. Show pending work separately. Revocations reduce current-effective counts but preserve historical acceptance flow.

12 Economic coordination and funding constraints

12.1 Current token-free conclusion

A product without a native token best suits Genesis: controlled tests on existing machines, with possible later per-task or support fees. This supports clear procurement and responsibility but still requires contracts, support, expense and refunds. Volunteers reduce cash outlay without eliminating economic cost or ensuring persistence.

Existing payment channels, invoices or suitable settlement assets may later meet real cross-participant payment needs. Selection depends on jurisdiction, counterparties, disputes, fees and custody. Avoiding a new asset does not remove price/credit risk or operational complexity. No channel or pricing is approved.

Every currently defined function works without a native token. Credentials establish access, queues and quotas distribute work, databases record acceptance/attribution and policies govern review. Future service payments can use agreed invoices or existing channels. A native token is neither a Genesis deliverable nor a default milestone. Reconsider it only if observed needs cannot be met by alternatives and value persists without appreciation expectations. Reward issuance that attracts supply is subsidy, not customer-demand evidence.

12.2 Functional alternatives

Task settlement pays for accepted service and requires delivery, price and dispute rules. Invoices or existing assets can serve the same purpose. A native asset adds pricing volatility and friction; customers may not want investment exposure to buy a service. Do not force every role to hold it.

A Node/Validator bond can constrain objectively provable breach only with evidence, appeal and correction. Alternatives include invitation, reputation limits, ordinary security arrangements or insurance. Slashing random-model disagreement or network failure can harm honest participants; stake does not prove correctness or independence.

Resource access allocates actual scarcity and requires observed congestion and demand. Queues, quotas, reservations and ordinary pricing are alternatives. Lockups without scarcity add friction and do not establish utility.

Compensation needs demand, a budget, acceptance and usage rights. Procurement by delivered result may pay under agreed terms; compensation for actual execution volume additionally needs execution and attribution evidence. Service revenue, grants, stated fees and nonfinancial credit are alternatives. Rewards exceeding consumption without a funding source remain subsidy.

Limited governance lets affected parties influence defined parameters under clear authority, upgrade and emergency rules. Maintainers, user representatives and contracts are alternatives. Token votes can concentrate power and cannot license others' data or guarantee safety. These functions do not prove issuance necessity. Any proposal should compare adoption, all costs, disputes and trust assumptions on the same workload without issuance subsidy or appreciation; no such evidence is supplied.

12.3 Unit-economic variables

Use a cohort with the same task family, scale, policy and observation period. Separate team demos, external free trials and paying customers. A is currently effective accepted primary tasks; exclude revoked and validation tasks. A is not billable count; disclose A_billable separately. X contains all primary execution attempts, including failures and retries. V contains all verification runs, including validation Tasks, internal checks, verification retries and corresponding human review. Never assign the same work to both X and V.

R is revenue for delivered services in that cohort under agreed terms, net of refunds and discounts. It is not all cash received that month. Exclude undelivered prepayments, financing and subsidies. Separately show R_cash, receivables and grants. Without supported price/payment terms, R is unknown; do not multiply free trials by a hypothetical price. Show unclosed obligations, disputes, refund exposure and unfinished tasks and update on closure.

c_x and c_v are average variable costs per execution and verification, although summing individual expenses is preferable for heterogeneous jobs. W is bandwidth, S storage, O otherwise uncounted coordination/support/disputes/rework, F reserved node capacity, and J payment fees and other uncovered fulfillment remediation. Use one currency, period and allocation rule, state tax treatment and mark unknown amounts. Refunds already deducted from R are not deducted again in J.

Unit full cost $C = (X * c_x + V * c_v + W + S + O + F + J) / A$. Average realized revenue $P = R/A$. Unit full-cost difference $m = P - C$ is not accounting net profit. If $A=0$, report totals and status without unit values. If R or a

material cost is unknown, do not state a definitive margin. When $A_{\text{billable}} < A$, allocation over all A remains necessary and the free-acceptance share must be disclosed.

The cash table uses actual payments and corresponding receipts. The economic-cost table additionally values founder/volunteer time and owned-device opportunity costs, with a basis. Hourly reserved capacity belongs in F , not again in c_x ; per-task compensation belongs in c_x . Prevent duplicate attribution of verification, wages and coordination. No measured costs, prices or payment evidence currently establish reasonable verification cost or commercial viability.

Capacity is an upper bound, not a sales forecast. Let H_x and H_v be available execution/verification capacity-time; t_x and t_v their mean busy time; u_x and u_v utilization; $a=A/X$ and $v=V/A$. Under stable scale and nonduplicated capacity, A cannot exceed $\min(H_x \cdot u_x \cdot a / t_x, H_v \cdot u_v / (t_v \cdot v))$, actual demand or human-support capacity. Split shared-machine time between H_x and H_v rather than counting an hour twice. If a or v is unstable, report observed throughput without extrapolation.

For each cohort report V/X , V/A , verification/execution expense ratio, verification expense/ A , p95 wait, and failure/revocation/rework rates. Lower acceptance and longer review queues worsen allocation of fixed cost. Verification costing more than execution is not inherently contradictory, but without users willing to pay for the evidence there is no basis for a low-cost-network claim.

12.4 Synthetic scenarios

Illustrative/synthetic data, not UNORYN measurements or an approved budget: a closed cohort has $A=A_{\text{billable}}=100$, revenue and receipts $R=120$ currency units, $X=120$, $V=110$, $c_x=0.20$, $c_v=0.30$, $W+S=5$, $O=20$, $F=20$. For simplicity only, $J=0$ with no refunds, open obligations or omitted labor; real accounting must verify these assumptions. Total cost is $24+33+5+20+20=102$. Unit cost is 1.02, revenue 1.20 and difference 0.18. Verification/execution expense is $33/24=1.375$; verification accounts for about 32.35% of cost. This is not profitability or acceptable-cost evidence.

Still synthetic: with the same fixed spending and utilization reducing accepted volume to 50, assume $X=60$, $V=55$ and $W+S=2.5$. Cost is $12+16.5+2.5+20+20=71$, or 1.42 per accepted task. Revenue of 1.20 then leaves a -0.22 difference. The assumption that support spending does not fall needs real checking.

Still synthetic: raising per-verification cost to 0.60 in the first scenario gives total cost 135 and unit cost 1.35. Users must value review enough to pay; otherwise compare batches, trusted hosted verification, fewer task families or stopping the service. Continuing reward issuance cannot conceal losses.

12.5 Cash, financing and exit

Token capitalization is a valuation based on price and quantity, not a bank balance. Trading volume is turnover, not protocol income. Sale proceeds become available funds only after actual transactions, expenses and obligations. Restricted pool assets, customer advances and refund reserves cannot simultaneously be unrestricted development cash.

Token name, ticker, chain, issuance, allocation, unlocks, entity, jurisdiction, TGE, exchanges and market-making remain undecided. This paper sets no FDV, fundraising target or return expectation. Legal advice must address actual activities, locations, flows and rights; an overseas entity with domestic development is not automatic clearance.

Without financing, use existing machines, manual approval, small packages and stoppable tests, expanding one variable at a time. Founders must confirm cash, weekly effort and a spending stop line. If essential engineering is unavailable, deliver local scripts and evidence formats instead of promising a public network. Preserve portable data and documents while stopping new long-term hosting/subsidy obligations when funding is inadequate.

13 Threats, security and data rights

13.1 Threat model

Attackers may be Nodes, requesters, verifiers, compromised dependencies or administrators. Objectives include unearned credit, exhaustion, data theft, evaluation contamination and history rewriting. A controlled prototype reduces exposure without eliminating attacks. The following controls are requirements with no measured adversarial results yet.

Threat	Objective and method	Genesis control	Residual risk
False identities	Many accounts/Nodes farm credit	Invites, device/control records, slot accounting	Declarations can lie; no permissionless Sybil defense
Copied answers	Claim execution credit without execution	Bounded accounting and commissions; controlled answer release	Inputs can be forwarded; correct output is not execution proof
Replay	Accept old work repeatedly	Attempt/generation binding, idempotency, unique keys	Implementation flaws or administrator bypass
Collusion	Nodes and verifiers cooperate dishonestly	Control disclosures, references and checks	A small team lacks institutional independence
False hardware/time	Inflate compensation or reputation	Diagnostic-only claims; capability tests	Challenges can be outsourced
Poor data	Poison labels or farm uploads	Permission/quality checks, counterexamples, versions	Labels and specifications can share errors
Benchmark leakage	Optimize known answers only	Access control, candidate history, new holdouts	Repeated tuning and similar-data leakage
Malicious tasks	Access files/network or exhaust resources	Data-only inputs, approved Runners, limits	Parser and isolation defects
Supply chain	Replace Runner or dependency	Frozen digests, dependency review, revocation	Trusted publisher compromise
Key compromise	Impersonate Node or administrator	Least privilege, rotation, revocation, audit	Historical impact may be unclear
Malicious Coordinator	Censor or rewrite acceptance	Exports, backups, action logs	Central control remains

Threat	Objective and method	Genesis control	Residual risk
Exhaustion	Large packages, long jobs, floods	Quotas, size limits, backoff, pause	Public operation needs dedicated stress tests

13.2 Controlled and public gates

A single-machine state test does not justify a general Runner for strangers. Controlled external testers need to see program content, resource limits, accessible paths, networking, retained logs and stop controls. Deny personal-file and credential access by default. No wallet key is required to bind a contribution.

Before public distribution, establish verifiable package provenance, signatures or equivalent authenticity, least privilege, enforced limits, task network rules, file isolation, version revocation, emergency stop and vulnerability response. Test malicious packages, failure to exit and unreachable revocation, not just normal startup. No arbitrary-code safety claim without independent review.

Docker packages environments and provides some isolation; it does not solve host permissions, mounts, networks or kernel flaws. Signatures establish package origin rather than harmlessness. Staking and votes do not protect the user's filesystem. Evaluate combined controls against actual privileges and failures.

13.3 Rights, retention and deletion

Inputs, models, outputs and logs each need license and visibility records. Requester permission claims require appropriate review. Isolate uncertain material rather than distribute it or include it in public benchmarks. Genesis samples exclude personal data; adding it requires a new access, retention and deletion design.

A task-level retention_policy distinguishes temporary debug logs, evidence artifacts and necessary audit records; durations require founder confirmation. Deletion must address accessible objects, caches, backup boundaries and invalidated references, not only remove UI links. Where audit and deletion needs conflict, retain only justified minimum metadata with explicit scope. Do not permanently anchor raw private material on-chain.

Public views reveal only authorized summaries and pseudonyms, not IPs, absolute paths, serial numbers, invites or keys. Hashing is not necessarily anonymization: low-entropy inputs may be guessed. Secrets and personal information should not be public merely because only their hashes are shown.

14 Governance, upgrades, disputes and recovery

Genesis uses maintainer governance with disclosed task, release, policy and pause powers. Two members may separate release and review responsibilities, subject to real availability. Disclose single-person dependencies; invent no independent committee.

Upgrades publish schema and compatibility changes and test old-client rejection or migration. Already dispatched Tasks retain frozen versions. Back up databases and test migrations on a copy before a maintenance window. Rollback must preserve or reconcile new records from that window; simply restoring an old backup may lose acceptance history.

Pause controls separately stop new claims, a task family, acceptance or a client version. Record reasons, affected objects and recovery conditions. Submission treatment follows task state, not ad hoc preference. Incidents require containment, evidence and scope review, then sanitized reporting; preserve failed records.

Disputes record the original decision, claim, evidence, receipt time, reviewer and outcome. Prefer a reviewer other than the original decision-maker; disclose if impossible. Review can uphold, revoke or request new work, never edit old artifacts to make the old decision appear correct. No automatic slashing is defined. Any future payment system needs separate disputed-funds rules.

Recovery drills cover database/process crashes, restart, restoration, missing artifacts and key compromise. Backups must reconcile database and artifact manifests and restore references, uniqueness and acceptance/contribution accounting. Use a database-supported backup method, not an unverified copy of a file being written. [R15] Recovery requires reproducible logs, not merely an openable file.

15 Preconditions for a more open network

Decentralization is not a calendar switch. First let external users export/check records, then introduce independently operated reviewers, and consider public audits of selective coordination. More complex communication, storage or settlement is justified only by observed limits of central control.

Open entry adds identity costs, junk tasks, malicious packages, copying, collusion and disputes. Invitation-based trust cannot be generalized into permissionless security. Challenges, trusted hardware, verifiable computation and multi-party review all have coverage, cost and trust limits; none is claimed adopted or effective without workload experiments.

Study anchoring only with genuine cross-entity audit needs, data minimization and affordable costs. Study P2P only for actual transport/reachability needs, not as an early commitment to arbitrary NAT, updates and endpoint security. Token necessity follows Section 12 and is not a decentralization score.

Training is a separate project addressing synchronization bandwidth, checkpoint rights, malicious gradients, fault tolerance and evaluation. Passing inference/classification tests does not solve training. A useful hosted evaluation product need not become a training protocol to preserve its narrative.

16 Experiments, gates and failure conditions

16.1 G0 Single-machine workflow

Hypothesis: one bounded task completes registration through acceptance. Minimum artifacts are Runner, reference verifier, database, CLI and JSON export. Prerequisites are authorized samples, frozen fields and resource limits. Normal, duplicate, incorrect-output, lease-expiry, cancellation-race and restart cases must satisfy the appendix, and acceptance/contribution counts must reconcile.

If it fails, do not add nodes. Narrow tasks, remove unnecessary states or fix idempotency. G0 needs no blockchain, graph database or public service. It tests basic semantics, not demand or malicious-participant security.

16.2 G1 Independent devices

Hypothesis: different real devices perform compatible tasks and handle disconnection. Artifacts are environment manifests, control relationships, runs and difference reports from 2-5 computers. Require G0, consent and supported environments. Every supported device needs successful and fault records with explained differences; Node IDs alone do not suffice.

If it fails, support only tested devices rather than count processes on one machine as many nodes. Network diversity supports reachability testing, not hardware independence. Two working computers do not prove scale, consensus or commerce.

16.3 G2 Real Dashboard and evidence

Hypothesis: readers can trace displayed results to real records. Artifacts are a read-only view and evidence export. Require actual database events; simulated data cannot masquerade as achievements. Arbitrary tasks must trace to inputs, attempts, verification and acceptance. Retries must not inflate success; revocation must change effective metrics; public fields must exclude secrets.

If it fails, return to CLI and exports until semantics are correct. Label synthetic demonstrations and separate them physically or logically from measured results. Visual polish is not acceptance evidence.

16.4 G3 External use

Hypothesis: non-team users gain repeat-use value. Artifacts are consented trials, authorized tasks, before/after comparisons and repeat-use records. Require safe controlled-test conditions. Interviewing 3-5 users and observing at least two distinct users returning without rewards may justify initial continuation, but task authenticity still needs checking.

If it fails, narrow to a local tool or reconsider the audience. Airdrop expectations, fabricated work and bought members do not repair demand evidence. Separate payment intent, revenue under terms and cash received.

16.5 G4 Cost and reliability

Hypothesis: fully verified service remains worth adopting. Artifacts include cash/economic cost tables by family, failure/support logs and acceptable price/time boundaries. Require real tasks and complete measurement. Reconcile the same cohort under Section 12.3, explain both cost views and recover key failures without duplicate accounting.

Commercial expansion additionally requires corresponding net service revenue to cover full economic cost, with sensitivity to measured failures, review queues and support. Grant-funded work can qualify only as a budgeted subsidized experiment with grant duration and a stop line, not commercial validation. Without evidence G4 stays unpassed.

On failure compare batch processing, trusted hosted verification, fewer families and local execution. Do not quietly weaken verification to make prices look lower. Stop paid expansion if adequate verification leaves no sustainable demand or funding.

16.6 G5 Openness and economics

Hypothesis: greater participation addresses observed supply, demand or trust limits. Artifacts are adversarial tests, independent security review, rights/dispute rules, economics and funding. Review open entry and payment choices separately. A new native-token proposal additionally needs evidence that existing settlement cannot meet the need. Founders must approve costs and rights.

Passing means security, demand, costs and activity-specific obligations are supportable, not reaching “10 nodes” or “200 members.” Otherwise remain controlled and token-free. Economic parameters need a separate decision, not promises made by this document's author.

16.7 Schedule and maturity

The handover targets end-of-September 2026 Genesis Ready and a small release October 1-3. As of this paper they are future targets, not commitments. If external-test gates fail, use the window for design review or an internal demonstration rather than force launch. Later references must not retroactively mark targets complete.

For a two-person team, proposed responsibilities are protocol/Runner and samples/verification/interviews, subject to confirmed capability and weekly effort. Defer independent research rather than attempt local AI, public clients, P2P, tokens and fundraising in one week. A complete paper is not proof of engineering resources.

17 Open questions and reference use

Unresolved questions include demand strength, tasks whose verification value exceeds cost, execution proof in open settings, independent operators, attribution/licensing disputes and maintenance capacity. They need not prevent internal controlled experiments but constrain external tests, public clients and economic activity differently.

The Litepaper and Whitepaper share terms, states and synthetic examples. Official sources describe mechanisms and their limits, not invented financing, prices, market statistics or endorsements. Existing provenance tools favor reuse; compute and incentive projects show why service, verification and economics each need evidence. The next engineering step is falsifiable acceptance tests, not copying design language into achievement claims.

Appendix A Minimum entity model

This is a proposed logical model, not a complete deployed database or SDK. Required fields establish relationships and constraints; implementation still needs full types, validations and migrations.

A1 Participants and devices

Contributor requires contributor_id, display_name, status and terms_version. Optional operator_group_ref privately records control relationships. Identity assurance is separate and defaults to SELF_DECLARED, not real-name verification. Node requires node_id, contributor_id, key_id, client_version, capabilities and status; device_ref and last_heartbeat_at are optional. Node and device relationships are not assumed one-to-one.

DeviceDeclaration records device_ref, operator_ref, os_arch, hardware_claims, evidence_level, verified_at and verification_method. Retain only necessary information. Public reports may describe two manually checked devices without publishing serial numbers. A device change creates a declaration version rather than silently reusing certification.

A2 Tasks and attempts

Task requires task_id, requester_id, task_family, task_role, spec_version, spec_digest, input_manifest_id, input_manifest_digest, runner_digest, verification_policy_id, verification_policy_digest, acceptance_policy_id, acceptance_policy_digest, accounting_policy_id, accounting_policy_digest, resource_limits, execution_deadline, review_deadline, max_attempts, state, visibility and license_refs. task_role is primary or validation; validation additionally requires verifies_task_id while primary omits it. Slot contains slot_id, task_id, role, generation, active_attempt_id and accepted_ref. Every initial slot.role is primary; statistics must use Task.task_role.

Attempt requires attempt_id, task_id, slot_id, node_id, contributor_id, key_id, generation, lease_expires_at, state and issued_at. Freeze contributor_id and key_id from authorized Node bindings at claim time, never from result payloads or later ownership edits. Optional fields are claimed_started_at, claimed_completed_at and failure_code. Only the assigned Node receives the secret lease token; store secure verification material and exclude it from public exports. New submission still requires current Node permissions.

A3 Artifacts and results

Artifact requires artifact_id, content_digest, media_type, byte_length, storage_ref, license_ref and visibility. Versioned claims may include derived_from and author_claims. Result requires result_id, task_id, slot_id, attempt_id, generation, task_spec_digest, input_digest, output_artifact_id, environment_digest, payload_digest and received_at. All redundant bindings must agree with frozen Task/Attempt. Result.task_spec_digest maps to Task.spec_digest; Verification.policy_id/digest maps to Task.verification_policy_id/digest. input_digest maps to Task.input_manifest_digest for the complete manifest, not concatenated sample content. result_id identifies a submission; output_artifact_id identifies what was submitted.

The environment manifest includes Runner/package digests, dependencies, OS/architecture, hardware claims and applicable model settings, with no secrets. Label self-reported cost/time as claimed, server measurement as observed and invoices as billed; one undifferentiated cost field cannot express their different assurance.

A4 Verification, acceptance and contributions

Verification requires verification_id, result_id, policy_id, policy_digest, verifier_id, verifier_version, review_round, job_generation, outcome, reason_code, evidence_artifact_id, started_at and finished_at, and also job_id. PASS/FAIL/ERROR are technical outcomes, distinct from Delta's four comparison outcomes. Multiple checks can exist, but the frozen policy defines aggregation.

Acceptance requires acceptance_id, task_id, slot_id, result_id, verification_set_ref, policy_version, policy_digest and accepted_at. Its policy fields refer to the Task's frozen acceptance policy. ContributionRecord requires contribution_id, acceptance_id, contributor_id, contribution_type, quantity, unit, accounting_policy_version and accounting_policy_digest, referring to the frozen accounting policy. Initial quantity=1; primary unit=accepted_primary_result_submission and validation

unit=accepted_validation_result_submission. Neither implies a price or compute duration. Actual-execution claims need separate evidence and review status; absent trusted observation, mark execution not independently verified. Knowledge/Intelligence remain license, author and adoption references, not four simultaneous economic credits from one Acceptance.

Revocation contains revocation_id, target_ref, reason, evidence_ref, actor_id and created_at; effective accounting joins those records. VerificationJob requires job_id, result_id, policy_id/digest, review_round, job_generation, assigned_verifier_id, lease_expires_at, state, retry_count and max_retries. Only the Coordinator atomically updates current rounds/generations. A result_id and a free-standing PASS are insufficient.

Artifact storage metadata distinguishes STAGED, READY and QUARANTINED and records references or retention pins. Only durable, digest-checked READY bytes can be linked to a Result. Garbage collection must not delete objects retained for pending review, effective acceptance or retention policy. One metadata transaction arbitrates deletion eligibility against reference creation; physical deletion follows authorization, with recovery scanning unfinished deletions.

A5 Connected illustration

Illustrative/synthetic data, not UNORYN measurements. The symbolic digest below is explicitly not a valid SHA-256 and is not an API request. The supplied JSON example separately contains a verifiable complete digest and synthetic=true.

```
{
  "synthetic": true,
  "schema_version": "0.1-proposed",
  "task_id": "demo-task-001",
  "task_family": "offline-rule-loop",
  "task_role": "primary",
  "slot_id": "primary",
  "attempt_id": "demo-attempt-002",
  "node_id": "demo-node-a",
  "generation": 2,
  "result_id": "demo-result-002",
  "output_artifact_id": "demo-artifact-a",
  "digest_symbolic": "SHA256_OF_CANONICAL_OUTPUT",
  "verification_id": "demo-verification-002",
  "verification_outcome": "PASS",
  "acceptance_id": "demo-acceptance-001",
  "contribution_id": "demo-contribution-001",
  "contribution_type": "Compute",
  "quantity": 1,
  "unit": "accepted_primary_result_submission",
  "economic_entitlement": null
}
```

The illustration represents a possible failed first Attempt followed by an accepted second Attempt in one slot. Two Attempts are not two accepted primary tasks. A real schema additionally checks Contributor consistency, foreign keys, time ordering and enums. Parsing JSON is not protocol verification.

Appendix B Proposed interfaces

The /v0 prefix is a design draft with no callable deployment supplied. Every write requires authentication, authorization, size limits and Idempotency-Key; GET has no write side effects. Task creation and Runner release are maintainer-only. A Node cannot submit a command string as a task.

Method and path	Role and request	Success	Important errors
POST /v0/nodes	Invite holder; capabilities/client version	201 Node and one-time credential	Invalid invite; unsupported version
POST /v0/nodes/{id}/heartbeat	Own Node; status/Attempt	200 server time	401 revoked; 403 unauthorized
POST /v0/claims	Node; families and free slots	200 lease; 204 no work	409 idempotency conflict; 429 limited
POST /v0/tasks	Maintainer; frozen TaskSpec	201 task_id	422 invalid package/permission/policy
GET /v0/artifacts/{id}	Authorized artifact reader	200 bytes and digest	403 forbidden; 404 absent
POST /v0/attempts/{id}/results	Assigned Node; bound Result	202 result_id and SUBMITTED	409 expired/stale; 422 invalid
POST /v0/tasks/{id}/cancel	Requester or maintainer	200 CANCELLED	409 already accepted; 403 forbidden
GET /v0/tasks/{id}	Authorized reader	200 state and visible evidence	403 or 404
POST /v0/verifications	Verifier service; fixed Result finding	201 verification_id	403 wrong role; 409 version conflict
POST /v0/revocations	Maintainer; reason/evidence	201 revocation	422 no basis; 403 forbidden
GET /v0/contributions	Participant or public view	200 paginated effective records	Unauthorized fields excluded

Acceptance may be an internal function, not a public route. Nodes cannot directly grant themselves credit. Claim retries return the original lease, even if it later expired, plus its current eligibility; they cannot silently allocate new work. An intentional new claim uses a new key.

Errors contain error_code, message, request_id, retryable and optional retry_after_seconds without sensitive stacks. Distinguish LEASE_EXPIRED, STALE_GENERATION, TASK_CANCELLED, PAYLOAD_CONFLICT, SCHEMA_INVALID, POLICY_MISMATCH, VERIFIER_UNAVAILABLE and STORAGE_UNAVAILABLE. HTTP status categorizes transport; stable codes guide client behavior.

202 means durably queued for verification, not Accepted. Permit result-status queries; after timeout retry the original key or query before creating duplicate work. Use stable list cursors and query timestamps so changing statistics do not silently double-count or omit rows.

Keep authentication out of URLs, public artifacts and logs. Do not retain registration secrets indefinitely as ordinary plaintext idempotency responses. Prefer a client-generated-key handshake or short-lived encrypted storage with an explicit recovery/reissue policy. Keys identify requests, not identity secrets. Revoked Nodes cannot replay old submissions through authorization checks; authorized participant read access can still retrieve history.

Appendix C Idempotency and transaction pseudocode

C1 Constraints

```
-- Proposed constraints, not a deployed migration.
UNIQUE(actor_id, route, idempotency_key)      -- request_dedup
UNIQUE(task_id, slot_id, generation)          -- attempts
UNIQUE(attempt_id)                            -- valid_results
UNIQUE(task_id, slot_id)                      -- acceptances
UNIQUE(acceptance_id)                        -- contribution_records
UNIQUE(result_id, policy_id, review_round)    -- verification_jobs
```

Acceptance uniqueness includes history. Revocation does not free the slot for a second historical Acceptance. Recompletion uses a replacement Task referencing the old one. Adjustments and Revocations correct accounting without creating a second initial contribution. Unique indexes are not a complete state machine; conditional field/state checks remain essential.

C2 Receiving a result

This is transaction pseudocode with unimplemented service helpers. Re-read eligibility, permission, pause and revocation in the same database write transaction. Administrative changes must also commit there rather than change only process caches. Normal key rotation can use a currently valid key belonging to the same assigned Node; historical ownership remains the Attempt snapshot.

```
def submit_result(actor, attempt_id, key, payload, lease_token):
    authorize_request(actor, attempt_id)
    digest = canonical_digest({"attempt_id": attempt_id, "payload": payload})
    # For a new result, bytes are already durable and immutable.
    with db.immediate_transaction():
        require_current_node_permission(actor, attempt_id)
        old = request_dedup(actor.id, "submit", key)
        if old:
            require(old.payload_digest == digest, "PAYLOAD_CONFLICT")
            response = old.response
        else:
            task, slot, attempt = load_bound_objects(attempt_id)
            prior = valid_result_for_attempt(attempt.id)
            if prior:
                require(prior.payload_digest == digest, "PAYLOAD_CONFLICT")
                response = original_submission_receipt(prior)
            else:
                require(task.state == "IN_PROGRESS", "TASK_NOT_ACTIVE")
                require_submission_controls_open(task, attempt)
                require_valid_lease_token(lease_token, attempt)
                require(slot.active_attempt_id == attempt.id, "STALE_GENERATION")
                require(payload.generation == attempt.generation == slot.generation,
                        "STALE_GENERATION")
                require(attempt.state in ["LEASED", "RUNNING"], "ATTEMPT_CLOSED")
                require_schema_bindings_and_durable_artifact(payload, task, attempt)
                received_at = server_now()
```

```

        require(received_at < attempt.lease_expires_at, "LEASE_EXPIRED")
        require(received_at < task.execution_deadline, "DEADLINE")
        result = insert_valid_result_unique(attempt.id, payload, digest,
                                           received_at=received_at)

        set_attempt_state(attempt, "SUBMITTED")
        enqueue_verification_in_database(result, task.verification_policy_id)
        response = submission_receipt(result)
        save_dedup_response(actor, "submit", key, digest, response)
    return response # every branch returns only after a successful commit

```

The idempotency digest covers attempt_id and canonical payload but excludes authentication secrets.

Reusing an actor's submit key with another Attempt conflicts instead of returning the other receipt. Check an existing Result before new-submission eligibility when the same Attempt/content arrives with another key, or SUBMITTED would incorrectly reject a valid retry. The original receipt is not current acceptance status; replay never requeues verification or restores closed eligibility. Changed content always conflicts. Current authorization still precedes replay. On uniqueness conflicts, roll back and recheck content rather than blindly report success.

Validate and durably write large immutable artifacts outside the transaction. Inside it, check prepared metadata, digests and bindings without downloading, scanning files or running verification. Bytes and retention pins cannot disappear between preparation and reference. Storage truth and durability remain trusted premises. Clean unlinked objects under policy; recovery must detect missing/corrupt referenced bytes and block acceptance. Files and the database have no common transaction, requiring crash tests.

C3 Acceptance

```

def accept_verified_result(result_id):
    authorize_internal_verifier()
    with db.immediate_transaction():
        task, slot, attempt, result = load_for_acceptance(result_id)
        old = existing_acceptance(task.id, slot.id)
        if old:
            decision = existing_decision_with_revocation_state(old)
        else:
            require(task.state == "IN_PROGRESS", "TASK_NOT_ACTIVE")
            require_acceptance_controls_open(task, attempt, result)
            require(server_now() < task.review_deadline, "REVIEW_EXPIRED")
            require(not revoked(result), "RESULT_REVOKED")
            require(result_is_valid_timely_submission(result), "INELIGIBLE")
            require(attempt.id == slot.active_attempt_id, "STALE_RESULT")
            require(result.generation == attempt.generation == slot.generation,
                   "STALE_GENERATION")
            require(attempt.state == "SUBMITTED", "INELIGIBLE")
            checks = load_current_policy_verifications(task, result)
            require_frozen_policy_satisfied(task, result, checks)
            require_ready_referenced_artifacts(result, checks)
            acceptance = insert_acceptance(task, slot, result, checks,
                                           policy=task.frozen_acceptance_policy)
            insert_contribution_unique(acceptance, attempt.contributor_id,
                                      task_role=task.task_role,
                                      policy=task.frozen_accounting_policy)
            set_slot_accepted_ref(slot, acceptance.id)
            append_audit_event("ACCEPTED", acceptance.id)
            set_task_state(task, "ACCEPTED")
            decision = acceptance
    return decision

```

controls_open checks the global and task-family acceptance switches, Task pause, nonrevoked Runner/policy/input/output versions, and current acceptance permission for both Node and claim-time Contributor. Stopping new claims alone does not forbid acceptance of submitted results, while pausing acceptance does. Suspected key-compromise windows must pause affected unaccepted results for review; rotating keys does not clear contamination.

Load checks from the database, never trust caller-supplied PASS or verification_set. Every check binds the same Result, frozen policy ID/digest, allowed verifier version and current review_round/job_generation, is not revoked, has valid verifier authority, and references saved evidence completed before review_deadline. Initial policy requires one effective PASS from the controlled reference verifier. Stale generations, FAIL, ERROR and unresolved conflicts do not satisfy it. Commissioned extra review is separate work and cannot vote away required checks or establish institutional independence.

Server eligibility time is sampled in the transaction and only matters after successful commit. Acceptance must itself reject expiry rather than rely on a scanner running first. Check Result, Artifact readiness and revocation; detected corruption freezes references immediately.

An existing Acceptance returns its own decision and current revocation status, not acceptance of a different Result. Ownership comes from immutable Attempt.contributor_id, never undefined result.contributor_id or client claims. Acceptance, slot, Task, contribution and audit commit or roll back together. This is database accounting consistency, not execution/payment proof or administrator honesty.

C4 Verification recovery

The durable verification-job table atomically owns review_round and job_generation; workers cannot choose them. Workers claim a finite job lease and verify fixed bytes outside a transaction. Completion rechecks assigned_verifier_id, current verifier permission, frozen policy, round, generation, LEASED/RUNNING state and server eligibility time strictly before both job lease expiry and Task.review_deadline. An unchanged generation alone is insufficient: expired but unreclaimed work cannot create an active PASS.

Duplicate valid completion returns the original receipt. Reusing a key with changed outcome or evidence digest conflicts. Stale work may leave diagnostics only. Condition the Verification write on the job generation so a late worker cannot override new work. Aggregate only policy-eligible current-round records. Recover worker crashes by requeueing; repeated ERROR blocks the Task without Node penalties.

Restart also scans staged/unlinked artifacts and accounting projections needing correction. Scans are idempotent and cannot create two tasks for one object. Recovery depends on durable facts, not a promise that messages never disappear.

Appendix D State transitions and metrics

Object and state	Event	Next state	Accounting effect
Task CREATED	Input/policy approved	QUEUED	No contribution
Task QUEUED	Valid claim	IN_PROGRESS	Attempt +1
Attempt LEASED	Start event	RUNNING	Telemetry only
Attempt LEASED/RUNNING	Timely valid submission	SUBMITTED	Result +1
Attempt LEASED/RUNNING	Expiry	EXPIRED	No effective contribution
Task IN_PROGRESS	Retryable execution failure	QUEUED	New generation pending
Result pending	Policy passes	Verification PASS	Distinct verified result
Result pending	Invalid content	Verification FAIL	Failure evidence retained
Result pending	Infrastructure error	Verification ERROR	Review retry; no Node penalty
Task IN_PROGRESS	ERROR limit or review expiry	BLOCKED	No automatic acceptance or redispach
Task BLOCKED	Approved recovery before review expiry	IN_PROGRESS	Verification only; no new contribution
Nonterminal Task	Cancellation commits first	CANCELLED	No later acceptance
Task IN_PROGRESS	Valid checks and acceptance transaction	ACCEPTED	Counts split by task_role
Unaccepted Task	Execution budget exhausted	FAILED	Logical failure +1
Effective Acceptance	Approved revocation	History retained; currently invalid	Effective accounting reduced

BLOCKED recovery requires a maintainer reason and time before the original review_deadline. Expired review permits diagnostics, cancellation or a replacement Task, not extended original eligibility. An Accepted Task does not return to QUEUED. Before redispach after FAIL, close the old Result's eligibility and record the reason so a later old verification cannot compete with the new generation.

Metrics include time range, family, role, policy version and effective/historical meaning. Control task scale/type across comparisons. Cumulative acceptance flow differs from current effective stock. Multiplying benchmark estimates by completed work does not yield total intelligence growth.

Appendix E Minimum acceptance matrix

These are system tests to implement, not tests already run during document editing. Retain inputs, versions, injected failures, database snapshots and assertions. Review 3 adds T37-T44. Attached document arithmetic checks do not establish that these system requirements pass.

ID	Fault or operation	Required observation
T01	Normal single completion	One Result, Acceptance and contribution
T02	Same key/content replay	Original response; no additional record
T03	Same key with changed payload or Attempt	PAYLOAD_CONFLICT; no overwrite or unrelated receipt
T04	Same Attempt/new key after submission or expiry	Authorized same content gets receipt without requeue; changed content conflicts
T05	Eligibility time exactly at lease expiry	Strict-before rule rejects; a timely committed response may arrive later
T06	Old result after a new generation	Even a correct old answer is ineligible
T07	Execution lease expires after durable submission	Continue review; no main-task redispatch
T08	Node disconnects before submission	Finite retries after expiry; history retained
T09	Submission response lost	Original key recovers result_id
T10	Cancellation races submission	Commit order governs; no new eligibility after cancellation
T11	Cancellation races acceptance	Cancellation first rejects; acceptance first makes cancel conflict
T12	Verifier crashes	Retry the same Result; no Node penalty
T13	Verification FAIL	Retain evidence; redispatch within budget or fail
T14	Duplicate verification completions	Only eligible generation counts; one acceptance
T15	Coordinator restart	Rebuild from DB without duplicate main credit
T16	Crash around DB commit	No partial Acceptance/contribution; queryable response
T17	Database busy or disk full	Explicit retry/error; no false success
T18	Interrupted or unlinked upload	No acceptance; orphan can be cleaned
T19	Invalid digest, schema or permission	Reject dispatch/submission without leaking evidence
T20	Path escape, oversized file or timeout	Enforce limits; no personal-directory access
T21	Unknown dependency or network request	Reject outside the allowlist

ID	Fault or operation	Required observation
T22	Revoked credential claim/retry	Reject new actions; preserve reviewable history
T23	Two Node IDs on one machine	Report one machine; no inflated independence
T24	Known shared implementation bug	Manually checked golden counterexample detects that bug, not every possible bug
T25	Leakage or incomplete predictions	Delta insufficient; no improvement claim
T26	Synthetic F1 and cost examples	Correct arithmetic and synthetic labels
T27	Derivation cycle or dangling reference	Reject; ordinary dispute references remain allowed
T28	Old slot after revocation	No second historical Acceptance
T29	Restore with missing artifact	Detect incomplete evidence; no reproducibility claim
T30	Dashboard reconciliation and redaction	Validation Task's primary slot is not customer demand; no secrets/private paths
T31	BLOCKED, acceptance pause and deadline race	Every entry refuses acceptance; recovery does not extend deadline
T32	Forged, stale or revoked PASS	Load current authorized database checks only
T33	Rebind Contributor after claim	Historical ownership unchanged; client cannot override
T34	Copying, forwarding and point farming	Record bypasses and false positives; valid output is not execution proof
T35	Threshold equality, missing/duplicate rows, zero resource baseline	Section 10.6 boundaries; insufficient first; no dropped failures
T36	Prepared bytes replaced or lost on restore	Detect and quarantine affected references; block acceptance

After the matrix, observe performance on real authorized tasks. A small fault matrix does not establish global scale, and failure to find a vulnerability does not prove absence. Public distribution still needs independent review. Expand tests with actual implementation instead of treating the written matrix as complete coverage.

E1 Additional boundary tests in this revision

ID	Injected case	Required observation
T37	Chinese text, CRLF, multiple/overlapping matches	Consistent UTF-8 byte offsets, sorting and frozen support rules
T38	Duplicate keys, nonfinite numbers, reordered rows	Reject invalid objects; canonical equivalent valid rows share a digest
T39	Same Node rotates key then retries; wrong token on new submission	No duplicate credit; wrong token and revoked credentials rejected
T40	Expired but unreclaimed verifier returns PASS	No change to effective review or acceptance; diagnostics only
T41	Acceptance/accounting policy changes mid-task	Old Task stays bound to frozen criteria and units
T42	Garbage collection races Result reference then crashes	Metadata arbitration preserves pinned objects; missing evidence blocks acceptance
T43	Threshold equality, invalid values, shared review cost	Exact rational boundaries; invalid values or allocation gaps are insufficient
T44	Revocation after export	Offline package states snapshot time; current query reports revocation

Appendix F Reference index

R1 is the Bitcoin whitepaper; R2-R3 cover GEODNET; R4-R5 Gensyn protocol and REE; R6 Bittensor Yuma; R7 W3C PROV; R8 OpenLineage; R9 MLflow; R10 Celery; R11 SQLite transactions; R12 RFC 8785; R13-R13b io.net; R14 SQLite foreign keys; R15 SQLite backups. The following links and accompanying source notes define dates and support boundaries. External precedents do not establish corresponding UNORYN capabilities.

Source links

Source check: September 12, 2026. Official documentation supports only the described mechanisms, not independent performance or UNORYN implementation. Version numbers identify the reviewed pages. See the accompanying source notes for boundaries.

R1 [Bitcoin whitepaper \(2008\)](#)

R2 [GEOD Token Introduction](#)

R3 [GEODNET station performance](#)

R4 [The Gensyn Protocol](#)

R5 [Gensyn REE v0.6.0](#)

R6 [Bittensor Yuma Consensus](#)

R7 [W3C PROV Overview \(2013\)](#)

R8 [OpenLineage Object Model 1.53.0](#)

R9 [MLflow Tracking](#)

R10 [Celery Tasks 5.6.3](#)

R11 [SQLite Transaction](#)

R12 [RFC 8785 \(2020\)](#)

R13 [io.net Proof of Work](#)

R13b [io.net Cloud Payments](#)

R14 [SQLite Foreign Key Support](#)

R15 [SQLite Backup API](#)