

Exact Rational Computation Exchange

A record-based comparison of message size, exactness and failure handling in a fixed deterministic suite

Author: Richard Janutka

Affiliation: Qonduit Limited NZ

Article type: Protocol measurement and verification study

Version: DAP-AI-PAPER-V2-20261016

Date: 16 October 2026

Publication status: Published in the Haivian Research Journal; public release authorized 16 October 2026

Canonical article: <https://haivian.com/research-journal/exact-rational-computation-exchange/>

Companion validation: <https://haivian.com/validation-journal/validating-exact-rational-computation-exchange/>

Rights: Copyright © 2026 Richard Janutka. Public article publication authorized; underlying code, data and private evidence are not released by this notice.

PUBLISHED ARTICLE

Canonical: <https://haivian.com/research-journal/exact-rational-computation-exchange/>

Companion validation: <https://haivian.com/validation-journal/validating-exact-rational-computation-exchange/>

Abstract

We compare a compact exact-rational encoding with a charged identity, revision and integrity adapter against a canonical typed exact-rational baseline with conditional compression and context reuse. Internally documented release records specify 24 scenarios in eight families, two arms and three deterministic repetitions: 144 scenario-arm-repetition cells and 378 raw subexchange rows. The original log records exact reconstruction and reference agreement for 348 harness-PASS rows, 30 typed fault detections and 18 permitted clean recoveries. A separate October 2026 audit independently calculated expected results using integer-pair arithmetic and matched all 348 archived result hashes, comprising 336 output successes and 12 correct first-false-obligation replies. Historical wire payloads were not archived for independent re-decoding. The prespecified secondary byte endpoint totals 86,490 candidate bytes versus 100,800 baseline bytes, a difference of -14.1964% under the declared simulated-traffic accounting rule. Repeated context reverses the difference: 37,578 versus 34,905 bytes, or +7.6579%. Every scored non-chain row already shares one coefficient denominator, limiting generalization of the size result. All 30 detection rows are untimed, a deviation from planned full-cost measurement. These results support a bounded local comparison and recorded-result verification, not general efficiency, cryptographic assurance, AI communication or product performance.

Keywords: exact rational arithmetic; message encoding; deterministic replay; fault injection; byte accounting; artifact verification

1. Scope and contribution

Exact computation exchange requires agreement on fraction normalization, ordering, schema, request identity, revisions and failure semantics. Exploiting mathematical structure may reduce payload size, while context and binding metadata can offset the reduction. We therefore compare complete declared request and response frames rather than inner payloads alone.

The contribution is a measurement of two concrete implementations on a fixed workload, with separate descriptions of the original instrumented observations, historical record review and later independent arithmetic audit. It is not a new rational representation or proof system. Existing exact, typed and schema-governed interchange provides relevant prior art [1-6], as do logical and cryptographic verification systems with different assurance models [7-9]. No exhaustive novelty or optimal-baseline claim is made.

The inspected measurement path generates local Python byte strings and contains no network transport or model-provider calls. The byte endpoint is simulated wire traffic, not measured network traffic. AI assistance in authoring and review is separate from the measured protocol workload. This v2 manuscript corrects reporting and interpretation of the unchanged study; it contains no new scored experiment.

2. Exact affine-rational semantics and admitted domain

Inputs are reduced rationals $x_i = p_i/q_i$, with positive denominators and zero represented by 0/1. Define

$$h = (1, x_1, \dots, x_n), \quad y_j = \sum_{i=0}^n a_{ji} h_i.$$

Each output row is a sparse ordered list of rational coefficients. With

$$z = (1, x_1, \dots, x_n, y_1, \dots, y_m),$$

each ordered obligation evaluates an integer affine form

$$g_k(z) = \sum_t c_{kt} z_{i_{kt}}.$$

Relation 0 requires $g_k = 0$; relation 1 requires $g_k \geq 0$. If every obligation holds, the result contains all reduced outputs in order. Otherwise it reports the zero-based index of the first false obligation. Such an obligation failure is a valid semantic result, distinct from protocol refusal of a malformed request. An empty form evaluates to zero. Within a contract, outputs depend on the constant and inputs; dependency-depth scenarios chain separate contracts using prior outputs.

The implementation accepts a bounded canonical domain, not arbitrary integer-affine contracts. Let $B = 2^{31} - 1$. The version is exactly 1, and the canonical contract object has exactly the ordered fields version, inputs, outputs, obligations. Integer-valued numeric positions require actual integers; booleans, floats, strings and null are not accepted as integers. Dimensions satisfy $0 \leq n \leq 16$, $1 \leq m \leq 16$ and $0 \leq k \leq 16$. Input and stored row-coefficient numerators lie in $[-B, B]$, and denominators lie in $[1, B]$. Fractions must be reduced. A row's common denominator and each scaled integer coefficient must also remain within B in magnitude, with the denominator positive. Stored sparse coefficients are nonzero and indices strictly increasing and unique within each row or obligation. Output-row indices range from 0 through n; obligation indices range from 0 through n+m.

Obligation coefficients are integers in $[-B, B]$. Nonempty obligation coefficient lists have gcd 1; an equality's first coefficient is positive. Empty rows and obligations are allowed. The common denominator for an empty row is 1. Duplicate input values, complete rows and complete obligations retain their positions; duplicate sparse indices within one row or obligation are prohibited. Semantic identity preserves version and all declared input, sparse-term, output-row and obligation positions. Equality of evaluated values alone is insufficient.

Native and outer frames are capped at 65,536 bytes, so fitting the semantic bounds does not waive framing limits. Candidate adapter revisions are integers from 0 through $2^{32} - 1$. Success-response numerators have magnitude at most $2^{1024} - 1$, and denominators lie from 1 through $2^{1024} - 1$, with canonical reduction. Every new chain stage must satisfy input admission again; unlimited chaining is not implied. These are implementation constraints, not sampled probability ranges.

3. Protocol implementations and assurance boundaries

3.1. Candidate

For a sparse rational row, the candidate uses $d_j = \text{lcm}_i(q_{ji})$ and stores $\alpha_{ji} = p_{ji}(d_j/q_{ji})$ as signed integers. Since $\alpha_{ji}/d_j = p_{ji}/q_{ji}$, the transformation is exact. The normalized representation requires $\text{gcd}(d_j, \alpha_{j1}, \dots) = 1$. Sparse index gaps use minimal base-128 variable-length encodings; signed values use ZigZag, an established technique [6]. Minimal encodings, the admitted bounds and normalization rules define canonicity.

The measured adapter includes state and request identities, revision, payload length and an unkeyed integrity checksum. Responses additionally bind the request hash. Its bytes are included in the reported totals. Freshness checking is an adapter function, not a property inferred from the inner encoding. Unkeyed hashes and checksums do not authenticate a sender or establish adversarial security.

3.2. Baseline

The comparator encodes exact rationals as canonical integer numerator/positive-denominator pairs in typed ASCII JSON. It selects deterministic DEFLATE only when shorter than the corresponding raw representation.

Frames include integrity, schema, operation, state, request and revision identities and response binding. Decoding checks canonical JSON and exact deterministic re-encoding, not merely syntactic JSON validity.

The baseline maintains distinct sender and receiver context objects. Context definition carries template, digest and inputs; updates carry inputs and revision/digest references; refreshes carry a replacement template. These messages are counted. The candidate sends the full contract at each exchange. Both arms use the same response executor, ordered semantics, scenario specification and recovery rights. This is a substantive typed/compressed/context comparator, not plain JSON; it is not established as optimal or strongest among all possible encodings.

3.3. Arithmetic checking and independence

In the original scored implementation, the response producer is separate from its expected-result evaluator. The evaluator does not produce arm responses. Both implementations nevertheless use Python `Fraction` [10] and the same affine specification; the baseline imports canonical JSON and contract validation from the oracle module. Thus independence means separation of response production from expectation calculation, with shared arithmetic-library, specification and validation dependencies. It does not mean independently originated truth or complete protection against common-mode errors.

Response decoders check canonical structure, binding, cardinality and failure-index range. They do not independently recompute the arithmetic or establish that an in-range false index is the first false obligation. Those properties are checked by the harness against the separate evaluator. The October audit's deliberately wrong, canonically encoded answer and later-false-index controls pass the native structural decoder and fail independent arithmetic. This is a boundary of the assurance mechanism, not an additional scored failure.

3.4. Prior art

RFC 8259 permits implementation limits on JSON numeric range and precision [1]; an application schema can nevertheless represent exact rationals. RFC 8949 specifies compact typed CBOR and deterministic encoding considerations [2]. The IANA registry includes the separately specified numerator/denominator rational extension, tag 30 [3]; this does not supply all reduction or rejection rules used here. Content MathML explicitly represents rational constants [4]. ASN.1 encoding rules and Protocol Buffers exemplify typed or schema-governed interchange [5,6], without establishing this study's particular rational semantics or byte performance.

Proof-carrying code uses checkable logical evidence of program safety [7]. Proof-carrying data and cryptographic verifiable computation study compliance or correct computation under explicit models and assumptions [8,9]. The present experiment implements neither these logical proof systems nor these cryptographic protocols. Exact arithmetic and hash agreement are not substitutes for their guarantees.

4. Documented design, endpoints and denominators

The historical internal release MXL-L1-SCORE-001 authorized one bounded scored invocation and prohibited scored reruns. Preserved records include an initial substantive reviewer REVISE, bounded corrections, development and reviewer fixtures, the original implementation package's v2 preregistration, release bindings and postflight records. That implementation revision predates and is distinct from this v2 manuscript. The records support an internally documented pre-execution freeze and report one scored invocation. Present-day hashes and timestamps do not independently prove original access isolation, chronology or absence of unlogged runs; no externally anchored registration or exhaustive process ledger is asserted.

The frozen hierarchy is retained here: **primary endpoints** are exact reconstruction, exact response or first-false index, fault/refusal detection and false acceptance. **Secondary endpoints** include complete declared message bytes, recovery and costs. Deterministic replay is a verification control. The v1 manuscript incorrectly called bytes and other endpoints primary; this is a reporting correction, not a change to the frozen study.

Quantity	Count	Interpretation
Scenarios / families	24 / 8	fixed workload cases and categories
Arms / deterministic repetitions	2 / 3	implementation comparison and replay checks
Primary execution cells	144	scenario × arm × repetition
Scenario-arm semantic cells	48	not 48 independently sampled tasks

Quantity	Count	Interpretation
Raw rows / rows per arm	378 / 189	stages, context, dirty attempts and recoveries included
Deterministic keys	126	three matching recorded identities per key

Per arm and repetition, the matrix expands to 12 ordinary rows, 14 dependency stages, 21 repeated-context exchanges, 10 changed/stale/replay rows and six dirty/recovery rows: 63 rows, giving $63 \times 2 \times 3 = 378$. The archived order matches the specified seeded shuffle and alternating arm order. A scenario is the paired experimental unit; subexchanges and repetitions are dependent. There is no population estimate, p-value or inferential effect claim.

Every scored non-chain row uses one common coefficient denominator within that row; chain rows use integer coefficients. The scored matrix therefore does not exercise heterogeneous coefficient denominators within a row. Generated inputs are positive, rows nonempty, and inequality obligations mostly single-output sign checks. These are material coverage limits, particularly for the size comparison.

Faults target corresponding failure functions using format-specific malformed objects. For example, the candidate malformed-response case truncates an actual native response, whereas the baseline uses float-numerator placeholder pairs. Such cases do not define equal corruption distributions, identical byte sizes or matched preparatory computation. Only F01-F03 permit one clean retry per arm/repetition; dirty attempts remain in the archive.

5. Original recorded observations and secondary byte results

5.1. Correctly assessed outcomes, faults and recoveries

The original log records 348 harness-PASS rows with exact-reconstruction and reference-match flags, and 30 typed detections. Harness-PASS means the instrumented assessment passed; it includes 336 successful output responses and 12 correct first-false-obligation responses. S03 first fails at index 3 and O03 at index 12, each across both arms and three repeats. O03's generator label `false_last` does not demonstrate a sole last-obligation failure: its final index is 15, but an earlier obligation fails. The separate v2 correction check further derived from the specified O03 input that obligations 12-14 are false while obligation 15 actually holds. That is a later arithmetic clarification, not an additional original observation. The fixture and logged first-false result remain unchanged.

All 18 permitted recoveries are recorded as passing after retained dirty attempts. No false acceptance or unexpected error is recorded in the fixed matrix. These are finite instrumented observations, not estimates of error rates or unseen-fault robustness.

Recorded typed detection	Rows
Stale prior revision at context update	3
Invalid native request	6
Invalid native response	3
Stale/future outer revision	3
Outer response identity/revision	3
Invalid request contract	6
Invalid fraction integer type	3
Mismatched response request ID	3
Total	30

The 30 rows represent ten fixed scenario-arm fault cases repeated three times. The archive retains flags, lengths and hashes rather than historical frame payloads; a later record review cannot directly re-decode those original exchanges. Section 6 distinguishes the later mathematical check from these original observations.

5.2. Complete declared simulated traffic

The prespecified secondary byte endpoint includes every recorded request/response frame and declared adapter overhead, including dirty attempts and clean recoveries. Source inspection confirms lengths are taken from the modeled frames. Under replacement-corruption accounting, a valid response constructed internally before creating its corrupted replacement is not an additional modeled transmission. Likewise, raw/compressed alternatives are internal preparation; only the selected baseline frame is counted. This is a declared traffic model, not all temporary memory or computational work.

Arm	Request bytes	Response bytes	Total bytes
Candidate	49,572	36,918	86,490
Baseline	54,285	46,515	100,800
Candidate minus baseline	-4,713	-9,597	-14,310

The relative difference is $-14,310/100,800 = -159/1120 \approx -14.1964285714\%$. The 174 candidate harness-PASS rows contain 35,772 native-request plus 10,614 request-adapter bytes, and 18,591 native-response plus 16,182 response-adapter bytes: 81,159 bytes in total. The 15 candidate dirty rows add 3,186 request and 2,145 response bytes, or 5,331; their native/adapter subdivisions were not logged. These figures reconcile to 86,490 without omitting dirty rows. Baseline dirty rows contain 4,056 request and 2,703 response bytes.

These totals are independently reproduced sums of archived length fields, supported by inspected counting paths. Missing historical payloads prevent independent direct frame-length measurement or re-decoding from the archive alone.

5.3. Family and scenario heterogeneity

Family	Candidate bytes	Baseline bytes	Difference	Relative to baseline
Short standalone	2,394	4,035	-1,641	-40.6691%
Rational width	3,054	4,701	-1,647	-35.0351%
Dependency depth	12,183	19,209	-7,026	-36.5766%
Malformed and recovery	8,157	10,923	-2,766	-25.3227%
Ordered obligations	4,305	5,589	-1,284	-22.9737%
Graph and row scale	5,217	6,195	-978	-15.7869%
Changed or stale assumptions	13,602	15,243	-1,641	-10.7656%
Repeated context	37,578	34,905	+2,673	+7.6579%

Displayed family percentages are rounded to four decimal places; byte counts are integer totals. Seven family sums favor the candidate; repeated context does not. Family averages also conceal adverse individual scenarios: G03 uses 2,892 versus 2,751 bytes (+5.1254%); R03 uses 27,651 versus 21,270 (exactly +30%); C03 uses 8,787 versus 7,953 (+10.4866%). These are parts of the same executed suite, not new experiments.

For the 63 candidate repeated-context records, native requests and responses total 27,876 bytes; the adapter adds 9,702, giving 37,578 versus the full baseline's 34,905. The adapter crosses that accounting threshold. It is not an isolated causal ablation: removing it removes matched functionality, while baseline context reuse, candidate full resend and jointly changing dimensions remain confounded. The grand total weights scenarios by message size and number of subexchanges. The homogeneous-denominator generator and chosen workload mixture preclude a general compactness ranking.

5.4. Descriptive timing and incomplete resource measurement

Timing exists for 174 harness-PASS rows per arm. Candidate wall time totals 557,651,900 ns, median 2,433,900 ns; baseline totals 578,500,900 ns, median 2,686,500 ns. The descriptive differences are -3.6040% and -9.4026%. They are not speedup estimates.

All 30 detection rows lack timing. This is a deviation from planned full-cost fault/refusal measurement; timing successful recoveries does not recover the preceding dirty-attempt costs. Host contention was uncontrolled.

Successful-row end-to-end measurements include reference checking and instrumentation/recording overhead up to the timer read, and are not isolated codec measurements.

CPU sums are 531,250,000 candidate ns versus 500,000,000 baseline ns. CPU is zero on 140/174 candidate and 142/174 baseline rows; nonzero row values are quantized at 15,625,000 ns. Comparative CPU inference is unsuitable. Maximum recorded Python allocation peaks are 23,103 and 331,090 bytes, respectively, but they are per-stage `tracemalloc` observations, not whole-process or simultaneous end-to-end memory peaks.

A post-child PowerShell reporting failure lost child exit code, exact wall time, CPU and peak working set. These remain UNKNOWN_NOT_INVENTED. The 2.2444644-second wrapper observation includes wrapper overhead and is not substituted. Complete measured resource-cap compliance cannot be established from policy limits, empty `stderr` or the complete row plan.

6. Evidence layers and the October 2026 audit

Four different evidentiary activities must remain separate.

1. **Original instrumented study.** The harness recorded reconstruction/reference flags, typed detections, result/frame hashes, lengths and partial timing. Its separate executor/reference modules share the dependencies described in §3.3.
2. **Historical read-only results review.** This recomputed recorded counts, byte sums, cross-arm result-hash equality and replay identities. It did not independently derive mathematical truth from the task inputs. Its PASS disposition was bounded by timing and telemetry limitations; it is not the source of the fresh arithmetic result below.
3. **Fresh audit, 16 October 2026.** A separately written checker transcribed the frozen input-generation specification and standard-library PRNG rules, then derived expected results using integer-pair arithmetic without the original Fraction-based oracle or scored harness. All 348 expected-result hashes match, covering 58 distinct scenario/revision task keys across two arms and three repetitions: 336 output successes and 12 correct first-false replies. This is conditional on the generation specification and accurate recording of result digests; it does not independently capture historical inputs or decode absent frames. Separate post hoc controls tested 134 small contracts, 1,027 signed-integer roundtrips and ten canonicity rejections without unexpected mismatch. The two structural-versus-semantic counterexamples in §3.3 are retained. These controls neither expand the original scored sample nor establish a new size effect. A distinct final reviewer reran the fresh checkers and obtained identical parsed results.
4. **This v2 revision.** Scientific wording, specification and evidence attribution are corrected against the audit. A separate correction reviewer checks the actual revised sources, with the reviewed byte identities recorded in the accompanying correction-check report. Such closure concerns these corrections; it is not a new experiment or blanket transfer of the v1 review's conclusions to v2.

The fresh audit matched declared lengths and SHA-256 digests for 10/10 result-manifest entries, 21/21 engineer-package entries and 5/5 results-review entries. The raw archive digest is 4e2b2caa8a86f85f200e32fd6258f69e7fe45479eb93082cf366168ff66fb965. These are integrity checks of supplied bytes, not authentication of authorship, prospective timing or scientific truth. The preserved failed review, corrections, fixtures and release bindings provide substantive internal lineage evidence beyond the hashes, with the historical limits in §4.

7. Interpretation and limitations

The recorded-result arithmetic and fixed-suite byte comparison are supported at the levels described above. The negative repeated-context family and adverse individual scenarios directly limit universal compactness claims. The separate arithmetic audit strengthens confidence in recorded outcomes without eliminating missing-frame or historical-custody limitations.

The experiment does not establish general correctness over all admitted contracts, unseen-fault detection, adversarial security, cryptographic soundness, sender authentication, cross-platform deterministic bytes or population error rates. It does not measure network performance, AI-to-AI communication, model reasoning, token use, monetary savings or production reliability. Fixed repeated tasks, narrow denominator/obligation structure, functionally rather than identically matched faults, the concrete baseline choice and incomplete timing preclude broader performance conclusions. No separate scientific observation was created to repair these limits in preparing v2.

8. Reproduction, authorship, interests and release

Two reproduction targets differ: recalculating published summaries from archived rows, and independently regenerating protocol frames and behavior. A **proposed minimum** release bundle would include the scenario matrix and generator specification; exact schemas, admission and canonicalization rules; fault/recovery and traffic-accounting definitions; evaluator; byte-exact fixtures or reference implementations; raw rows and available payloads; analysis/checker programs; exact software versions; commands and a hash ledger. Its sufficiency would require an actual independent exercise. Fixtures alone do not prove general implementation behavior, and missing historical frames cannot be restored by a checklist. The article and its public publication package alone are not sufficient for independent reproduction.

Richard Janutka is the research author, affiliated with Qonduit Limited NZ. Haivian Technology is the corporate author of a separate validation manuscript. Qonduit Limited NZ and Haivian Technology share common ownership; that validation is not external institutional review. The scientific audit used separately dispatched AI contexts with independent initial assessments, directed challenge and a distinct final reviewer. This is procedural AI separation within a shared project/provider environment, not external human peer review or a guarantee against shared biases.

Human-directed AI systems assisted source inspection, calculation/checker authoring, literature checking and drafting. No AI model appears in the inspected measured protocol path. The author remains responsible for the manuscript and disclosures. Prose and figures prepared for the project are not, by that fact alone, independently cleared for originality, ownership or release. Publication of this article is authorized. Contributor authority, third-party/dependency provenance, and code/data/prose licensing for any underlying materials remain separate. No legal or patent-clearance conclusion is offered. This article release does not authorize public distribution of underlying code or evidence.

9. Conclusion

In the unchanged fixed suite, the recorded candidate frames total 14.1964% fewer bytes than the specified typed baseline under the declared simulated-traffic model. Repeated context costs 7.6579% more, with a 30% penalty in its largest scenario. In the later independent arithmetic audit, independently calculated expectations match all 348 archived observed-result digests, while historical frame reconstruction remains unverified. The revised account supports this bounded comparison, preserves the adverse observations and measurement deviations, and makes no general efficiency, proof-system, AI-communication or product claim.

References

1. Bray T, ed. *The JavaScript Object Notation Data Interchange Format*. RFC 8259. December 2017. <https://www.rfc-editor.org/rfc/rfc8259.html>
2. Bormann C, Hoffman P. *Concise Binary Object Representation (CBOR)*. RFC 8949, STD 94. December 2020. <https://www.rfc-editor.org/rfc/rfc8949.html>
3. IANA. *Concise Binary Object Representation Tags*, tag 30, rational number. <https://www.iana.org/assignments/cbor-tags> . Tag specification: <https://peteroupc.github.io/CBOR/rational.html> . Accessed 16 October 2026.
4. W3C Math Working Group. *Mathematical Markup Language (MathML) Version 3.0*, second edition. 10 April 2014. <https://www.w3.org/TR/2014/REC-MathML3-20140410/> . Rational constants: chapter 4, §4.2.1.1.
5. ITU-T. *Information technology — ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. X.690 (02/2021). <https://www.itu.int/itu-t/recommendations/rec.aspx?lang=en&rec=14472>
6. Google. *Protocol Buffers: Encoding*. <https://protobuf.dev/programming-guides/encoding/> . Accessed 16 October 2026.
7. Necula GC. Proof-carrying code. *Proceedings of POPL '97*. 1997:106-119. <https://doi.org/10.1145/263699.263712>
8. Chiesa A. *Proof-carrying data*. M.Eng. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science. 2010. <https://dspace.mit.edu/entities/publication/6cf95695-f0ce-468e-8d8f-16e6033511e9>

9. Gennaro R, Gentry C, Parno B. Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. *Advances in Cryptology – CRYPTO 2010*, LNCS 6223. 2010:465-482. https://doi.org/10.1007/978-3-642-14623-7_25
10. Python Software Foundation. *fractions — Rational numbers*. Python 3.13 documentation. <https://docs.python.org/3.13/library/fractions.html> . Accessed 16 October 2026. Documentation reference, not an assertion of the original execution's Python patch version.

Evidence availability

The underlying release records, raw archive, generator, implementations, historical reviews and fresh-audit materials remain private. The publication package preserves the checked v2 source identity and a release hash ledger, but it does not publish private audit evidence, code or historical wire frames. Consequently, public reproduction remains limited in the ways stated in Sections 6 and 8.