

The Respondent-Judge Protocol

Running Theory-Guided Probes Systematically, with Content and Software Verification

J. Konstapel, Leiden, 17 October 2026

© 2026 J. Konstapel, Constable Research, Leiden. All rights reserved. Quotation with attribution is permitted; reproduction of the whole requires written permission.

Version 1.0

0. What this document is for

This document turns a working method into a protocol. With it, one assignment is enough to start a full run. Claude Code executes the run. The vacuum.net theory steers it. Every result is tested twice: on its content, and as software.

The protocol grew out of three rounds of probes on open mathematics, run in October 2026.

- **Round one** hunted for counterexamples and found nothing.
- **Round two** steered by closure and produced 61 verified theorems.
- **Round three** tested a meta-pattern on 178 probes. It yielded an exact formula for closed trit windows and a closed form for covering periods. It also found one structural line through four families: multiplicative closure is exactly decidable; additive closure is not.

Each rule below exists because something in those rounds went right or went wrong. Section 9 lists the lessons one by one.

The document has three layers.

1. **The guidance layer** (Sections 2–3). How the theory turns into a finite, ordered list of probes.
2. **The verification layer** (Sections 4–7). Who does what, which gates a result passes, and how content and software are tested.
3. **The reporting layer** (Sections 8–10). How results become essays, appendices and external review packets.

Appendix A holds the assignment itself, ready to copy. It is self-contained.

1. Philosophy

1.1 Closure first

The theory has one selection rule (axiom N2): a configuration persists only if it closes on itself in phase. In the image of the fishing net, a mesh holds where its strands meet.

Applied to mathematics, the rule sets the target. A probe looks for what closes, and for the rule every closure obeys. It does not look for the place where a conjecture breaks. A search for an exception to a true statement returns nothing. A search for the structure of what closes returns structure, and structure can be proven.

1.2 Why a respondent and a judge

The occasion was the WIRED AI Lab newsletter of 14 October 2026. Two of its findings shaped the protocol.

- An MIT study (Thompson, Manso, Mayrink Verdun) measures AI on theory problems from unseen papers. It separates a **respondent**, who gets only the question, from a **judge**, who holds the answer keys and a list of traps.
- Work from Meta and Duke (Hu et al. 2026) shows that agents learn to pass a test without solving the task, and pass the trick on. A rule checked by the code that produced it learns its own test.

Two conclusions follow. The one who checks must hold what the one who answers lacks. And a prediction counts only if it is fixed before the count, out of reach of the counting code.

1.3 The machine computes, the human attests

Claude Code is the executor of every generation, count, proof attempt, test and check in this protocol. No step is written in the passive voice without that executor.

The machine computes and derives. It does not decide. The one human act is attestation: reading the essay and judging whether a result stands.

1.4 Status words

The protocol uses the status vocabulary of the theory's formal statement, extended by three words the probes needed.

Status	Meaning
proven	follows by a complete proof given in the record
computed	checked exactly in a stated finite range, by a stated algorithm
input	taken from a named source and used as given
corresponded	compared with a published result, with the numbers stated

Status	Meaning
open	stated as a problem, not settled
known	already in the literature; the source is named
refuted	a counterexample is given and independently re-checked

“New” is not a status. It is a note meaning: not found in the literature that was searched.

2. The guidance layer: from theory to probes

The theory does not prove the theorems. It limits the search space and orders it. This section states how each part of the theory becomes a search operator.

2.1 Axioms as search operators

N1 (Strand). Closures are self-crossings. Every distinction is the strand touching itself. In arithmetic, a closure is a place where an orbit, a sum or an expansion returns to itself. Generator: enumerate returns. Cycles of maps, repeated values, exact identities, periodic digit windows.

N2 (Closure). Look for the rule of what closes. Generator: enumerate all closures of a family in a range. Order them by their rest, the distance to exact closure. Find the rule all closures obey. Prove it.

N3 (Memory). Orientation matters. The net keeps the orientation of what was laid down first. Generator: run every family in both orientations and both signs. a versus $-a$, $+1$ versus -1 , $a^x - b^y$ versus $b^y - a^x$, both signs of a covering (Brier). In round three, sign variants exposed a whole missing identity type.

N4 (Context). Constants are state functions. A number that looks like a constant is expected to be a function of a local state. Generator: for every constant found, search for the state variable it depends on. Examples from the rounds:

- the minimal covering period is a function of the order profile of the base;
- the size term of the trit-window count depends only on $p \bmod 54$;
- the coupling γ is fixed by the return probability P_d of a free path (Theorem 10).

N5 (Scale). The same winding at every depth. Generator: every rule found is re-tested at another depth. Other bases, powers of the base, negative bases, larger ranges, other moduli. A rule that holds at one depth only is recorded as computed, not as a law.

2.2 Theorems and lemmas as instruments

Theorem 4 (nine of twenty-seven). The closing trit windows form the kernel of the trit-sum homomorphism, a subgroup of index 3. Instrument: in any balanced-ternary statistic, shifting by one third moves the window through the three cosets. This gave the one-in-three lemma and removed the conductor-3 characters from the trit-window formula.

Theorem 5 (two kinds of closure). A window closes iff its entries are all equal or all different. Instrument: refine every window count into homogeneous and heterogeneous closures. This dimension was not yet probed. It belongs in the next matrix.

Lemma (closure probability). For independent entries with law p on $\mathbb{Z}_3$,
 $P(\text{close}) = (1/3)[1 + \hat{p}(\omega)^3 + \hat{p}(\omega^2)^3]$.

Instrument: the baseline for every window count. A deviation from it measures dependence between entries. In the digits of a/p the dependence $r_{\{i+1\}} = 3r_i - d_i p$ produced the exact size and phase terms.

D9, Corollary 9.2, Theorem 10. For a counted density, the closing energy is the Cramér rate function, and $\gamma = -\mu\kappa_3/\kappa_2^2$ exactly. For a knot on a free path, $\gamma = -(1 + P_d)/P_d$. Instrument: for any family of counts, predict γ from the cumulants before measuring it.

2.3 The two returns

Every closure built from windings of small bases has two measures.

- **Size.** How close b^n comes to a^k . The convergents of $\log_b a$.
- **Phase.** When b^n returns modulo m . The order $\text{ord}_m(b)$, and the characters that can block a return.

The probes predict with these two measures first. Round three showed that size mostly acts as a window, a budget or a completeness bound. Phase carries the exact rules.

2.4 Routing by kind of closure

Round three produced a routing rule. Before a family is probed, Claude Code classifies its closure equation.

Kind	Form	Expected result	Strategy
Multiplicative	one relation: is $-u_1/u_2$ in a subgroup of $(\mathbb{Z}/q)^*$?	exact rule by orders and characters	predict exactly; aim for a proof
Additive	a sum of three or more units vanishes mod q	expected counts; exact islands	intensity model; then hunt for exact regimes ($q = 1$, Cauchy-Davenport)
Algebraic	a polynomial identity	closes for a whole family	identify and remove the families first;

Kind	Form	Expected result	Strategy
			probe the sporadic rest
Integral	an exponential equation in $\mathbb{Z}$	solved case by case	Baker plus lifting the exponent; prove completeness of each range

This table saves work. It says in advance where a proof is likely, where a statistic is the best possible, and where a family must be removed before anything else is seen.

3. The probe matrix: all possibilities, in order

“All possibilities” means a finite, explicit list. Claude Code builds it as a matrix and runs it in priority order.

3.1 The axes

1. **Family.** The closure families so far:
 - rest between windings ($2^n - 3^k$, Pillai, Catalan-type);
 - exact closure (unit fractions, Erdős–Straus);
 - prime windings (coverings, Sierpiński and Riesel, primitive divisors);
 - cycle maps ($x/2$ or $(ax + b)/2$, m-ary maps);
 - digits and trit windows (balanced ternary of a/p , digit maps);
 - scale and orbits (Kaprekar, Ducci, lonely runners, Gleason polynomials);
 - never-closing windings (irrationality measures, elliptic curves);
 - counted densities on paths and lattices (γ , Theorem 10).
2. **Kind** of closure: multiplicative, additive, algebraic, integral (Section 2.4).
3. **Theory hook:** the axiom, theorem or lemma that motivates the probe (Section 2.1–2.2).
4. **Operator:** base change, sign change, power base, negative base, modulus change, range extension, refinement by Theorem 5.
5. **Measure tested:** M’s specific size measure, M’s specific phase measure, or generic algebra only.
6. **Range tier:** T0 small (seconds), T1 medium (minutes), T2 large (up to an hour), T3 record (hours, only after T1 succeeds).

3.2 Building the list

Claude Code takes the product of the axes and removes the empty or meaningless cells. Each remaining cell becomes a probe with an identifier, for example `cyc-add-N5-negbase-T1-017`.

The matrix is written to `matrix.csv` before any probe runs. It is the plan. A coverage report at the end shows which cells ran, which were skipped, and why.

3.3 Priority

Claude Code orders the probes by expected information, using four rules.

1. Multiplicative cells first, because they are most likely to yield a proof.
2. Cells adjacent to an earlier result next. N5 says the rule should hold at the next depth.
3. Cells near the frontier found in round three: the zone $\gcd(q, a - 2) > 1$, three runs with moderate q , base -2 .
4. T3 record runs only after the T1 run of the same cell has produced a rule.

3.4 Parallel lines

The matrix is split by family into parallel lines. Each line is run by its own respondent agent. A line reports when its cells are done or its budget is spent.

4. Roles

Each role is a separate agent or a separate script. No role reads code written by a role that checks it, or is checked by it, until its own work is fixed.

Role	Executor	Produces	Must not
Coordinator	main Claude Code session	matrix, assignments to lines, synthesis	write probe code
Respondent	one agent per line	rules, proofs, records	see the judge's code
Predictor	separate script, called by the respondent	predictions/<id>.json + sha256	read any enumeration
Enumerator	separate script	enum/<id>.json	run without the prediction file
Scorer	separate script	hits, misses, false alarms	change predictions
Line checker	separate script, exact arithmetic	per-probe check	share imports with the enumerator
Independent judge	separate agent per family	verdict report	read respondent code before its own tests
Literature agent	judge or separate agent	sources, novelty note	rely on memory alone for "new"
Recomputer	coordinator, new code	third computation of key results	reuse any earlier code
External reviewer	another AI or a mathematician	review of the appendix packet	—
Attestor	the human	judgement of the essay	—

The coordinator handles the external review itself. It prepares the packet, collects the review, and works every point into the next version.

5. The probe lifecycle: nine gates

A probe moves through nine gates. Each gate leaves a file. A probe that skips a gate cannot reach the status proven.

G0 — Registered. The probe has a row in `matrix.csv`: family, kind, theory hook, operator, measure tested, range tier.

G1 — Predicted. The predictor script writes the prediction rule and the predicted list to `predictions/<id>.json`. Its sha256 goes into the record. The prediction uses only the measures named in the matrix row.

G2 — Enumerated. The enumerator refuses to start unless the prediction file exists and its hash matches. It writes every closure in the range to `enum/<id>.json`, with the range and the completeness argument.

G3 — Scored. The scorer compares the two files. It reports hits, misses and false alarms, and lists every miss and false alarm explicitly.

G4 — Ruled. If the score is perfect, the respondent states the rule as a theorem. If not, the respondent characterises the misses exactly. A miss is information: it shows where the measures are insufficient.

A rule found on data is a **fit**. It is labelled as a fit and re-tested on a fresh, disjoint range through G1–G3 again.

G5 — Proven or bounded. The respondent writes the proof step by step, naming every hypothesis and every gap. Or it states the exact finite range in which the rule is computed.

G6 — Line-checked. A separate script with exact arithmetic re-checks the closures and the rule.

G7 — Judged. The independent judge of the family tests the claim with its own code on a fresh range, then reads the proof, then searches the literature. It issues one verdict per claim:

- VERIFIED;
- VERIFIED-WITH-FIXES (the fixes are listed);
- REFUTED (with a counterexample);
- GAP (the missing step is named);
- KNOWN-ALREADY (with the source).

G8 — Recomputed. For every result that enters an essay as a main theorem, the coordinator recomputes it a third time with new code.

The status in the final record follows the gates: proven needs G5 to G7 passed without open gaps; computed needs G2 to G6 at least.

6. Content verification

Content verification asks whether the mathematics is right, whether it is new, and what it says.

6.1 The proof check

The judge reads every proof against this list.

1. **Statement.** Are all quantifiers explicit? Are all objects defined?
2. **Hypotheses.** Is each hypothesis used? Is each use justified? Typical failures: a gcd condition left out ($\gcd(q, a - 2) = 1$ in round three); a condition $0 < a < p$ silently assumed; a “sorted” order lost when $a > p$.
3. **Converse.** Is an “iff” proven in both directions? Round three refuted two converses.
4. **Edge cases.** Small primes, p dividing a parameter, exponent 0, negative bases, perfect-power bases.
5. **Cited lemmas.** Is each cited result stated exactly as in its source? Is it used within its hypotheses?
6. **Worked example.** Does the proof come with at least one fully computed instance?
7. **Re-derivation.** The judge re-derives every non-trivial step independently and records the result.

6.2 Fresh-range testing

The judge picks its own test ranges, disjoint from the respondent’s. It writes them down before running. Typical choices: the next block of primes, bases the respondent did not use, a random sample of unused parameters.

6.3 Literature and novelty

The literature check has four layers.

1. **Classical sources** for the family. Named per family in the matrix: for example Rav, Mordell and Elsholtz–Tao for unit fractions; Steiner, Böhm–Sontacchi and Simons–de Weger for cycles; Bennett, Scott and Styer for Pillai.
2. **Recent preprints.** arXiv searches with the key objects and the exact formula shape.
3. **Tables.** OEIS, erdosproblems.com, known record lists.
4. **Method check.** If the method is known, the result is “new as an instance”. If the result itself is found, the status is known.

“New” is written only after all four layers. Memory alone never counts.

6.4 The specificity test

For every rule the judge asks: does this test the theory’s specific measure, or would the same shape appear for any sum over a coset? Round three found that

a split into “size term plus phase term” is the orthogonality decomposition of any coset sum. Such a result confirms algebra. The record says which it is, in the field `m_specific`.

6.5 Framing claims

Claims about what a result means are checked like theorems. In round three the line between two and three terms was first called the line between Baker’s theory and the abc conjecture. The judge showed this was off by one term and modular. Interpretive claims go through G7 too.

7. Software verification

Software verification asks whether the code computes what it claims, completely and reproducibly.

7.1 Independence

- Respondent, enumerator, line checker and judge use separate code. No shared imports between a producer and its checker.
- The judge writes its code before reading the respondent’s code.
- Where possible, the checker uses a different algorithm from the enumerator. For example: a divisor enumeration against a residue-type dynamic program; a C enumerator against a Python one; an exact solver (CP-SAT) against a closed form.

7.2 Exact arithmetic

- Integers and rationals are exact: Python integers, Fraction, sympy, or PARI/GP where available.
- Floating point is allowed only for character sums and Fourier coefficients. Then the record states the tolerance and the largest deviation found.
- Every reported identity is re-checked in exact arithmetic before it is written.

7.3 Completeness

A range is complete only with a written argument.

- **Bounds.** State the bound used (for example Laurent 2008 for linear forms in two logarithms) with its constants.
- **Every value.** Check the bound for every value of the running variable, not the first that works.
- **Continued fractions.** Use the denominators q_n of the convergents, and their multiples where needed.
- **Least-element bounds** for cycles: compute the bound and record it.

- **Timeouts.** A timeout is recorded as GAP. It is never counted as “no solution”.

7.4 Tests the code must pass

Each line keeps a test suite. Claude Code runs it before every count that enters a record.

1. **Golden tests.** Known values the code must reproduce. Examples: Bennett’s list of Pillai exceptions; Mordell’s classes; the known cycles of $3x + 1$ and $5x + 1$; the 16 exceptions of $b^n - 1$ without a primitive divisor; the covering table for small bases.
2. **Differential tests.** Two independent algorithms on the same input must agree exactly.
3. **Metamorphic tests.** Symmetries the result must respect: negation invariance, coset invariance, $a \mapsto -a$ with the matching sign change, base b against b^2 .
4. **Property tests.** Random inputs with a checked invariant (for example: every enumerated cycle satisfies its closure equation $x \cdot D = b \cdot S$ exactly).
5. **Non-vacuity tests.** The checker must fail on a planted error. A dynamic program that never finds a failure is suspect until it finds one where one exists (round three used $a = 19$ for this).
6. **Boundary tests.** Exponent 0, p dividing a parameter, the smallest primes, the edges of the range.

7.5 Provenance

- Every file carries a sha256 in `manifest.json`, with its creation time and the script that wrote it.
- The enumerator checks the prediction hash before it runs.
- Every run is deterministic. Random samples use a recorded seed.
- The counting unit is defined in the record: ordered tuples or distinct identities. Round three reported 1001 ordered tuples that were 853 distinct identities.
- One command, `make verify`, re-runs all checks of a line from the raw files and compares hashes.

7.6 Blind spots

When generator and brute force are written by the same agent, they can share a blind spot. In round three both missed the opposite-sign identity type until an independent census of triples exposed it. Rule: every family gets at least one census by a method that does not use the respondent’s classification.

8. Scoring and continuation

8.1 Scores

Every probe reports:

- hits, misses, false alarms;
- $\text{recall} = \text{hits} / (\text{hits} + \text{misses})$;
- $\text{precision} = \text{hits} / (\text{hits} + \text{false alarms})$;
- the full lists of misses and false alarms.

A blind test is a test whose predictions were hashed before the enumeration and whose range the predictor never saw. Only blind scores are reported as predictive power.

8.2 When to continue

The coordinator continues a line while it produces at least one of these:

- a rule with a perfect score on a fresh range;
- a proof that closes a gap;
- a characterised miss that suggests a new rule;
- a frontier cell (Section 3.3) that moves.

It stops a line after a fixed budget of probes without any of these. It reports the stop and its reason.

8.3 When to stop the round

The round ends when every line has stopped and every judge has reported. Then the coordinator synthesises. An essay comes only after the research is complete and ordered.

9. Lessons built into the protocol

Round	What happened	Rule
1	237 probes hunted for counterexamples; none found	closure first (Section 1.1)
2	a Pillai bound stopped at the first value that worked	check every value (7.3)
2	a continued-fraction routine used numerators	use denominators q_n (7.3)
2	one rule refuted by 7/261	judge tests before reading (5, G7)
3	prediction and count in one script, same second	separate hashed predictor (G1-G2)
3	two exact rules fitted on data, then called predictions	label fits; re-test fresh (G4)
3	a failed “iff” relabelled as its surviving half	failures stay recorded (G4)

Round	What happened	Rule
3	an identity type missed by generator and brute force	independent census (7.6)
3	a shortcut for $s = 0$ caused 142 misses	boundary tests (7.4)
3	two-run rule exact only if $\gcd(q, a - 2) = 1$	hypothesis check (6.1)
3	“three or more terms only statistical” too strong	converse and regime check (6.1)
3	boundary first framed as Baker versus abc	framing claims are judged (6.5)
3	“period 9 never occurs” refuted at $b = -32409$	judges extend ranges (6.2)
3	1001 tuples were 853 identities	define the counting unit (7.5)
3	size-phase split was generic algebra	specificity test (6.4)

10. Reporting

10.1 Files per line

- `matrix.csv` rows of the line;
- `<line>.jsonl`, one record per probe (schema in Appendix B);
- `<line>-summary.md`;
- `proofs/`, `predictions/`, `enum/`, `tests/`, `manifest.json`;
- `judge-<line>.md` with the judge’s own code in `judge-<line>/`.

10.2 The essay

Each round is reported as an English essay, not as a report. It has:

- the philosophy behind the round;
- the occasion;
- the experiment;
- the evaluation and the adjustment;
- an annotated reference list, with “Why read?” and reading advice.

It opens with what was found. Corrections are stated briefly and plainly. It carries a version number and the copyright line, and it never names the author personally. Sentences stay under about forty words. It is delivered as Markdown and as a typeset PDF.

10.3 The appendix

The main theorem of a round goes into an appendix, complete and self-contained:

- setting and notation;
- lemmas with proofs;
- the theorem;
- the proof, step by step;
- at least one worked example with all numbers;
- the checks, with ranges and largest errors;
- the literature comparison.

10.4 The external packet

The appendix doubles as the packet for external review. Respondent and judge are the same model. They catch slips and code errors, as the rounds showed, but not a blind spot of the model itself. An outside reviewer can. The coordinator processes every point the reviewer raises and issues a new version.

Appendix A. The assignment

This text can be given to Claude Code as it stands. It assumes no earlier context.

Assignment: a theory-guided probe round with respondent-judge verification

Suggested approach. Claude Code judges what fits the situation.

Goal. Run a systematic round of computational probes on families of arithmetic closures. Find, for each family, the rule every closure obeys. Prove the rules where possible. Verify every result on content and as software. Report the round as an English essay with an appendix holding the main proof.

Theory guidance. The search is steered by the vacuum.net theory. Its relevant content:

- *N2 Closure.* A configuration persists only if it closes on itself in phase. So look for what closes and its rule, not for counterexamples.
- *N3 Memory.* Orientation matters. Run every family in both signs and orientations.
- *N4 Context.* Constants are state functions. For every constant found, find the state variable it depends on.
- *N5 Scale.* The same winding appears at every depth. Re-test every rule at another base, power, sign, modulus or range.
- *Theorem 4.* Of the 27 balanced-trit windows (w_1, w_2, w_3), exactly 9 close ($\text{sum} \equiv 0 \pmod{3}$). They form a subgroup of index 3.
- *Theorem 5.* A window closes iff its entries are all equal or all different.
- *Closure lemma.* For independent entries with law p on $\mathbb{Z}_3$: $P(\text{close}) = (1/3) [1 + \hat{p}(\omega)^3 + \hat{p}(\omega^2)^3]$, with $\hat{p}(z) = p_0 + p_1z + p_2z^2$ and $\omega = e^{(2\pi i/3)}$.

- *Two returns.* Closures of windings of small bases have a size measure (how close b^n comes to a^k ; convergents of $\log_b a$) and a phase measure ($\text{ord}_m(b)$; characters that block a return).
- *Routing.* Classify each closure equation first. Multiplicative (one subgroup relation mod q): expect an exact rule. Additive (three or more units summing to $0 \bmod q$): expect counts, look for exact regimes such as $q = 1$ or Cauchy–Davenport. Algebraic (polynomial identity): remove the families first. Integral (exponential equation in $\mathbb{Z}$): Baker plus lifting the exponent, with proven completeness.

Step 1. Build the matrix. Families $\times$ kind $\times$ theory hook $\times$ operator (base, sign, power, negative base, modulus, range, Theorem 5 refinement) $\times$ measure tested $\times$ range tier (T0–T3). Write `matrix.csv` before running anything. Order by priority: multiplicative cells first; cells adjacent to earlier results; frontier cells; record runs only after a medium run gave a rule.

Step 2. Run parallel lines, one respondent agent per family. For every probe:

1. A separate predictor script writes `predictions/<id>.json`. Record its sha256.
2. A separate enumerator refuses to run without that file. It writes all closures in the range, with a completeness argument.
3. A scorer reports hits, misses, false alarms, with full lists.
4. A perfect score becomes a theorem with a proof. Otherwise, characterise the misses exactly. A rule found on data is a fit: re-test it on a fresh range.
5. A separate line checker re-checks in exact arithmetic.

Step 3. Software standards. Exact arithmetic for all integer and rational work. Every bound checked for every value. Continued fractions by denominators q_n . Timeouts recorded as GAP. Each line keeps golden, differential, metamorphic, property, non-vacuity and boundary tests and runs them before each count. Every file in `manifest.json` with sha256. Deterministic runs with recorded seeds. One `make verify` per line. At least one census per family by a method independent of the respondent’s classification. Define the counting unit.

Step 4. Independent judges, one agent per family. Each judge writes its own code first, tests the main claims on fresh ranges it picks itself, then reads the proofs line by line, then searches the literature (classical sources, arXiv, OEIS and problem lists, method check). One verdict per claim: VERIFIED, VERIFIED-WITH-FIXES, REFUTED, GAP or KNOWN-ALREADY. Judges also check interpretive claims and whether a result tests the theory’s specific measures or only generic algebra.

Step 5. Recompute every main theorem a third time with new code.

Step 6. Continue a line while it yields rules with perfect fresh scores, closed gaps, characterised misses or a moving frontier. Stop it after a fixed budget without these, and say why.

Step 7. Report once all lines and judges are done:

- an English essay: philosophy, occasion, experiment, evaluation and adjustment, annotated references with “Why read?” and reading advice; opening with what was found; short sentences; version number; the copyright line “© 2026 J. Konstapel, Constable Research, Leiden. All rights reserved. Quotation with attribution is permitted; reproduction of the whole requires written permission.”; no personal mention of the author;
- an appendix with the main theorem: setting, lemmas, theorem, full proof, worked example, checks, literature;
- Markdown and a typeset PDF;
- a coverage report of the matrix;
- the appendix doubles as a packet for external review.

Status words. proven, computed, input, corresponded, open, known, refuted. “New” means not found in the literature searched.

Appendix B. The probe record

One JSON line per probe. Fields:

Field	Content
id	probe identifier, matching matrix.csv
family, kind, hook, operator, tier	the matrix row
m_specific	size, phase, both or generic
range	the exact range, and the counting unit
prediction_file, prediction_sha256	from G1
enum_file, enum_sha256, completeness	from G2
hits, misses, false_alarms, recall, precision	from G3
miss_list, fa_list	explicit lists, or a file reference
rule, fit_or_prediction	from G4
proof_file, gaps	from G5
line_check	script name and outcome (G6)
judge_verdict, judge_fixes	from G7
literature, novelty_note	sources searched and found
recomputed	for main results (G8)
status	one of the status words
timestamps	per gate

Appendix C. Judge brief

Each judge receives this brief, filled in for its family.

You are the independent judge for the family F . Work in your own directory. Write your report to judge- F .md.

Before reading any proof or code of the respondent, write your own code and test the main claims below on ranges you choose and record first. Then read the proofs line by line, re-deriving each step. Then search the literature.

For each claim give one verdict: VERIFIED, VERIFIED-WITH-FIXES, REFUTED, GAP or KNOWN-ALREADY, with numbers, counterexamples and sources.

Check also: the protocol traces (hashes, file times, order of prediction and count); the counting unit; the completeness bounds; whether each result tests the theory's specific measures or generic algebra; every interpretive claim.

Compute everything yourself. Keep each run under a stated time; record what you could not finish.

Claims: *list*.

Appendix D. Verification checklist

Content

- ☐ statement complete, quantifiers explicit
- ☐ every hypothesis used and justified
- ☐ both directions of every "iff"
- ☐ edge cases: small primes, divisibility, exponent 0, signs, powers
- ☐ cited lemmas exact and within hypotheses
- ☐ worked example with all numbers
- ☐ fresh-range test by the judge
- ☐ four-layer literature check
- ☐ specificity test recorded
- ☐ interpretive claims judged

Software

- ☐ producer and checker share no code
 - ☐ exact arithmetic; float tolerances stated
 - ☐ completeness argument with constants
 - ☐ every value of the bound checked; q_n used
 - ☐ timeouts recorded as GAP
 - ☐ golden, differential, metamorphic, property, non-vacuity, boundary tests pass
 - ☐ independent census per family
 - ☐ manifest with sha256; prediction hash checked by the enumerator
 - ☐ deterministic, seeds recorded
 - ☐ counting unit defined
 - ☐ make verify passes from raw files
-

Annotated references

Knight, W. (2026). “AI’s mathematical prowess was just the beginning.” *WIRED AI Lab* newsletter, 14 October 2026.

Why read? The occasion of the protocol: the MIT study, the Meta-Duke transmission result and the Harvard-NTT population threshold in one issue.

Manso, G. (2026). “A Benchmark for Scientific Reasoning: Frontier AI on Unseen Science.” *OpenAI Forum*, 21 July 2026.

<https://forum.openai.com/public/videos/event-replay-a-benchmark-for-scientific-reasoning-frontier-ai-on-unseen-science-2026-07-21>

Why read? The source of the respondent-judge design: premises only for the respondent, answer keys and traps for the judge. Reading advice: the transcript, minutes 2 to 9.

Hu, Y., Ghahremani, P., Aghajani, R., Pasupuleti, M. and Bhattacharjee, H. (2026). “Reward Hacking as a Transmissible Strategy in Multi-Agent Coding Systems.” *arXiv:2610.12822*. <https://arxiv.org/abs/2610.12822>

Why read? Why predictions are hashed before counting and why judges write their own code. A test visible to the generator is learned. Reading advice: Sections 3 and 5.1, Theorem 5.1.

Konstapel, J. (2026c). “The Vacuum.Net Theory: A Formal Statement.” *Version 3*, 16 October 2026.

Why read? The axioms N1–N5, Theorems 4, 5 and 10, the closure lemma and the status vocabulary that the protocol uses. Reading advice: Sections 0, 1, 4 and 7.

Konstapel, J. (2026d). “Closure First: Probing Open Mathematics with the Rule of the Vacuum.Net.” *Version 1.0*, 16 October 2026.

Why read? Rounds one and two: why counterexample hunting failed and closure-first worked. The origin of the six-step probe.

Konstapel, J. (2026e). “Two Kinds of Closure: A Meta-Pattern Tested on 178 Probes.” *Version 1.1*, 17 October 2026.

Why read? Round three: the routing rule by kind of closure, the protocol tightening, and Theorem A with its full proof as a model appendix.

Nosek, B. A., Ebersole, C. R., DeHaven, A. C. and Mellor, D. T. (2018). “The preregistration revolution.” *Proceedings of the National Academy of Sciences* 115, 2600–2606.

Why read? Preregistration in the empirical sciences: fixing the prediction before the data. Gate G1 is the same idea, made checkable by a hash.

Claessen, K. and Hughes, J. (2000). “QuickCheck: a lightweight tool for random testing of Haskell programs.” *Proceedings of ICFP 2000*, 268–279.

Why read? The origin of property-based testing, used in Section 7.4.

Chen, T. Y., Kuo, F.-C., Liu, H., Poon, P.-L., Towey, D., Tse, T. H. and Zhou, Z. Q. (2018). "Metamorphic testing: a review of challenges and opportunities." *ACM Computing Surveys* 51(1), article 4.

Why read? How to test code whose correct output is unknown, through relations the output must satisfy. The basis of the symmetry tests.

McKeeman, W. M. (1998). "Differential testing for software." *Digital Technical Journal* 10(1), 100-107.

Why read? Two independent implementations on the same input as a test of both. The basis of producer-checker separation in software.

Laurent, M. (2008). "Linear forms in two logarithms and interpolation determinants II." *Acta Arithmetica* 133, 325-348.

Why read? The standard bound for completeness of exponential ranges. Reading advice: the main theorem and its corollaries.

Bourgain, J., Glibichuk, A. and Konyagin, S. (2006). "Estimates for the number of sums and products and for exponential sums in fields of prime order." *Journal of the London Mathematical Society* 73, 380-398.

Why read? The theory behind the additive row of the routing table: why closures of three or more units mod q admit average counts but rarely exact rules.