

October 19, 2026, Ken Racen

A white paper on separating calculation from explanation in AI systems for measurement science.

Abstract

Large language models can explain measurement science well, but they generate numbers rather than compute them, and their outputs are not reproducible from run to run. Metrology's foundations (*traceability, validated calculation software under ISO/IEC 17025, the GUM's defined uncertainty procedure, and agreed decision rules*) assume a calculation that yields a single answer per input. This paper proposes an architecture in which deterministic code computes every reported value, using exact rational arithmetic wherever a value can be exact, and the language model explains only the final results. It applies the architecture to SI unit conversion, GUM uncertainty budgets, Welch-Satterthwaite effective degrees of freedom, TUR and TAR, and conformity decision rules with guard bands, and works through the GUM H.1 gauge-block budget stage by stage. A case study of eight validated calculators on the MetTutor® platform reports 68,000 independent random test cases with no mismatch in any displayed value. The approach is consistent with published research on delegating computation from language models to programs, with the NIST AI Risk Management Framework and the OECD AI Principles, and with metrology's established guidance on software validation. Its main limitation is that the model-written explanation remains probabilistic and can be tested only by sampling.

1. Introduction

Large language models are now used to teach and support measurement science. They explain the GUM fluently, describe decision rules, and walk a technician through an uncertainty budget. They are not reliable calculators. A language model produces each token from a probability distribution learned from text; it has no arithmetic unit, and the same prompt can return different numbers on different runs.

In metrology, this matters more than in most fields. A measurement result is a value with a stated uncertainty, traceable to the SI through an unbroken chain of calibrations, and every number in it must be reproducible by an assessor. A result that is right most of the time is not a measurement result. A combined standard uncertainty that is wrong in its third significant figure can move a reading across a guard band and change a pass into a fail.

The problem is not hypothetical. During the development of MetTutor®, a skill that asked the model to compute an uncertainty budget, including a Welch-Satterthwaite evaluation, reported a combined standard uncertainty of 0.002788 V, while the hand-verified value was 0.002822 V. The error was 1.2 %, plausible on its face, and nothing on the screen showed it was wrong.

This paper advances a fundamental architectural principle: in AI systems for metrology, deterministic code performs all numerical calculations, while the language model interprets and explains the results. It distinguishes deterministic, probabilistic, and generative computation and explains why metrology standards require deterministic numerical processing. Using the **MetTutor®** platform as a case study, the paper addresses calculator validation, evaluates the approach's benefits and limitations, and considers its implications for current research and policy.

2. Deterministic, probabilistic and generative computation

A **deterministic algorithm**, given a particular input, always produces the same output; its current state fully determines its next state. Computer science identifies ways an algorithm loses this property: it depends on external state beyond its input, is sensitive to timing, or a hardware fault changes its state unexpectedly (Wikipedia, "Deterministic algorithm").

The three broad categories of AI systems and their key distinctions are:

- **Deterministic Systems:** Operate according to predefined rules and logic. Given the same inputs and operating conditions, they consistently produce the same outputs.
- **Probabilistic Systems:** Use statistical models to estimate the likelihood of different outcomes based on patterns learned from data. Rather than relying solely on fixed rules, they generate predictions or classifications based on probabilities. Many machine-learning classifiers and forecasting models fall into this category.
- **Generative Systems:** Use learned patterns and probabilistic models to generate new content, including text, images, and computer code. Large language models (LLMs) are a prominent example. Rather than simply retrieving existing information, they generate outputs based on patterns learned during training.

Two technical facts sharpen the distinction for numerical work.

First, a **language model's numbers come from learned, approximate mechanisms, not from a specified and validated algorithm**. NIST's Generative AI Profile lists "*confabulation*" among the twelve risks it identifies: the production of confidently stated but erroneous content, which it describes as a natural result of models that generate outputs approximating the statistical distribution of their training data. It adds that such models sometimes give logical steps for how they reached an answer even when the answer itself is wrong (NIST AI 600-1, 2024). In metrology, that combination is the worst case: a wrong uncertainty presented with a plausible derivation.

Second, even a language model set to its most repeatable setting is not reliably deterministic in practice. Researchers at Thinking Machines Lab showed that LLM inference services return different outputs for the same prompt at temperature zero and traced the main cause to kernels that are not batch-invariant: server load changes the batch size, which changes the order of floating-point reductions. Because floating-point addition is not associative, summing the same numbers in a different order yields a different result. They also showed that batch-invariant kernels make inference reproducible (He et al., 2025). Reproducibility alone does not settle the matter, however: a reproducible model that computes a wrong number returns the same wrong number every time.

The case for deterministic calculators rests on correctness that can be validated, not merely on repeatability. The non-associativity of floating-point arithmetic, described in Goldberg's classic survey, is why the calculators in this paper use exact arithmetic wherever possible (Goldberg, 1991).

Property	Deterministic computation	Probabilistic model	Generative language model
Same input, same output	Always	Same distribution; a sampled output may differ	Not guaranteed, even at temperature 0
How a number is produced	Computed by an explicit algorithm	Estimated from learned parameters	Generated token by token from learned text patterns
Error behavior	Systematic and reproducible; found by testing	Statistical; characterized by validation metrics	Confabulation: fluent, plausible and wrong
Can be validated case by case	Yes: one correct output per input	Only statistically	Only by sampling; a pass does not carry to the next run
Explains its result	Only what it was programmed to state	Weakly	Fluently, in natural language
Role in AI-assisted metrology	Compute every reported number	Not used for reported values	Explain, teach and guide

The table points to a division of labor rather than a choice between camps. Industry sources reach the same conclusion: **determinism does not mean eliminating language models but building systems in which each component serves the role it is suited to**. The rest of this paper applies that idea to metrology.

3. Why metrology requires determinism

Every requirement that gives a measurement result its authority assumes that the calculation behind it can be repeated exactly. ***Metrology's governing documents do not use the word "deterministic", but each of them depends on it.***

Traceability is a chain of calculations. The VIM defines metrological traceability as the property of a result that allows it to be related to a reference through a documented, unbroken chain of calibrations, each contributing to the measurement uncertainty. Each link carries a computed value and a computed uncertainty. If an arithmetic step cannot be reproduced or is not documented, the chain is broken at that point, however good the physical comparison was.

Software that calculates results must be validated. ISO/IEC 17025:2017 requires that a laboratory's information management systems used for collecting, processing, recording, and reporting data be validated for functionality before introduction and again after changes. Validation (VIM 2.45) requires evidence that the software is adequate for its intended use; a central part of that evidence is verification that it produces the correct output for defined inputs, which is only meaningful if the output for a given input is fixed. A component whose output varies from run to run can be sampled, but it cannot be validated in this sense.

Uncertainty evaluation is a defined procedure. The GUM (JCGM 100:2008) specifies how standard uncertainties are combined, how effective degrees of freedom are found, and how a coverage factor is chosen. The GUM fixes the method but leaves some choices open, such as how to round and whether to interpolate or truncate v_{eff} . Once those choices are documented, two competent people applying the GUM to the same model and the same inputs must obtain the same combined standard uncertainty. The procedure is then an algorithm, and its output is not a matter of opinion.

Conformity decisions follow an agreed rule. ISO/IEC 17025:2017 requires the decision rule to be agreed with the customer (7.1.3) and stated with any statement of conformity (7.8.6). ILAC-G8 and JCGM 106 define those rules in terms of acceptance limits, guard bands, and risk. Applying a stated rule to stated values must give one answer. A decision that could come out differently on a second run is not the result of a rule.

Requirement	Source	What a non-reproducible calculation breaks
Unbroken, documented chain to the SI	JCGM 200:2012 (VIM) 2.41; ISO/IEC 17025:2017 6.5	The link whose value cannot be recomputed
Software validated before use and after change	ISO/IEC 17025:2017 7.11	Validation itself: there is no fixed output to compare
Uncertainty evaluated by a defined procedure	ISO/IEC 17025:2017 7.6; JCGM 100:2008	Agreement between two evaluators of the same budget
Decision rule agreed and applied	ISO/IEC 17025:2017 7.1.3, 7.8.6; ILAC-G8; JCGM 106	The conformity statement: one rule, one answer
Results reviewable by an assessor	ISO/IEC 17025:2017 7.5, 7.8	The audit trail from input to reported value

These requirements describe a property of the calculation, not of the person or system that presents it. A language model can present a result without weakening any of them, provided it receives the result already computed.

4. A deterministic architecture for AI metrology

The architecture draws a single boundary: every number is generated on the code side, which is deterministic and validated, while the language model sits on the other side and receives only finished results. Figure 1 shows the five stages a request passes through in the compute-then-explain architecture, with 5 stages and 2 zones.

Code computes every number; the model only explains

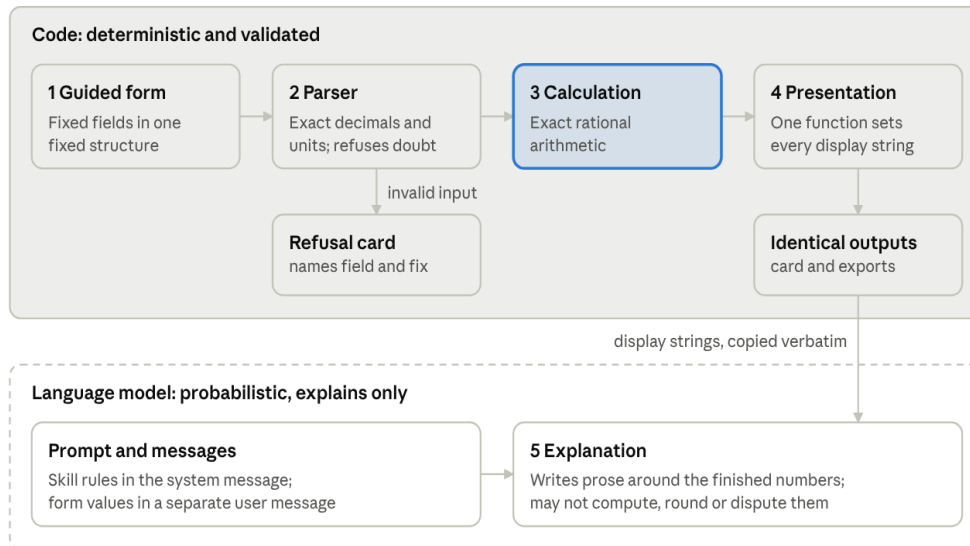


Figure 1. Invalid input stops at the parser with a refusal; a valid request is computed exactly, and the model receives only the display strings the card already shows.

1. **Guided form.** The user enters values in fixed fields. The form composes them into one fixed text structure, so every request arrives in the same shape, and nothing depends on how the user phrased a question.
2. **Parser.** Code reads each field and converts it to an exact quantity with its unit. Anything that cannot be read unambiguously (a comma used as a decimal mark, a missing unit, a value out of range) is refused with a card that names the field and the fix. The parser never guesses.
3. **Calculation.** Code computes the result from the decimals the user typed. Where a result can be exact, the arithmetic is rational: values are held as fractions of integers, so $0.1 + 0.2$ is exactly 0.3 rather than the binary 0.30000000000000004. Square roots and truncations are decided by integer comparison, so a last digit is never moved by rounding error.
4. **Presentation.** One function turns the results into every display string: the headline, the card, and each export. Because the card, the PDF, the Word report, and the spreadsheet use the same strings, they cannot disagree.
5. **Explanation.** The language model receives the skill's rules as its system message, the user's values as a separate message, and the computed results in a block marked as verified and authoritative. It writes the teaching text around those numbers. It may not recompute, re-round, or dispute any of them, and every number it writes must appear in the block.

Three design rules hold the boundary in place:

- **Refuse rather than estimate.** An input the parser cannot read exactly produces a refusal, not a plausible answer. A wrong number that looks right is the failure this architecture exists to prevent.
- **Exact where possible, validated where not.** Square roots are determined exactly by integer comparison. Only transcendental quantities (factors involving π , normal-distribution risk integrals, exponentials) are computed in IEEE-754 double precision within a single controlled server environment, and each is checked against an independent high-precision reference.
- **Instructions and data travel separately.** Placing the user's values inside the model's instructions makes the model unable to tell its rules from its inputs. Keeping them in separate messages is a structural requirement, not a style choice.

The result is a deterministic system, as required. **Given the same form, the server computes the same card on every run, and every device displays it identically.** The explanation may vary in wording from one run to the next. It is instructed to state only numbers from the computed block, and live tests check that it does so, but this cannot be guaranteed for every run (Section 8.2).

5. Applications in metrology

Each core calculation in a calibration laboratory is a fixed procedure with a single correct output, and each has a specific way of going wrong when it is approximated. This section takes five of them in turn and names, for each, the error a deterministic implementation removes.

5.1 SI units and conversion factors

Since the revision of the SI that took effect on 20 May 2019, the seven base units are defined by fixing the numerical values of seven defining constants, such as the Planck constant $h = 6.626\,070\,15 \times 10^{-34}$ J s; those values have no uncertainty (CGPM, 2018; NIST SP 330). Many customary units are in turn defined exactly in terms of SI units: the inch has been exactly 25.4 mm since 1959 (NBS, 1959), the pound exactly 0.453 592 37 kg, the thermochemical calorie exactly 4.184 J.

A conversion factor derived from exact definitions is itself exact, but it is often not a terminating decimal. The pound-force is exactly $0.453\,592\,37\text{ kg} \times 9.806\,65\text{ m/s}^2 = 4.448\,221\,615\,260\,5\text{ N}$. The pound-force per square inch is that force divided by $(0.0254\text{ m})^2$:

$$1\ \text{psi} = \frac{4.448\,221\,615\,260\,5\ \text{N}}{(0.0254\ \text{m})^2} = \frac{8\,896\,443\,230\,521}{1\,290\,320\,000}\ \text{Pa} = 6\,894.757\,293\,168\ldots\ \text{Pa}$$

The denominator contains the factor 127^2 (from $25.4 = 127/5$), so the decimal never terminates. **A deterministic converter stores each factor as the exact fraction implied by its definition and rounds only once, at display.** A language model recalls a factor from text it has seen, using whatever precision that text used, and may recall it differently next time.

5.2 Uncertainty evaluation under the GUM

For a measurement model $Y = f(X_1, \dots, X_N)$ with uncorrelated input estimates, the GUM's law of propagation of uncertainty gives the combined standard uncertainty:

$$u_c^2(y) = \sum_{i=1}^N c_i^2 u^2(x_i), \quad c_i = \frac{\partial f}{\partial x_i}$$

Each standard uncertainty $u(x_i)$ comes from a Type A evaluation (the experimental standard deviation of the mean, $s/\sqrt{n}$) or a Type B evaluation (a stated bound a divided by $\sqrt{3}$ for a rectangular distribution, $\sqrt{6}$ for triangular, $\sqrt{2}$ for U-shaped, or an expanded uncertainty divided by its stated coverage factor). The expanded uncertainty is $U = k \cdot u_c$.

The procedure is fully specified, so two competent evaluators of the same budget must agree. Where they disagree in practice, the cause is almost always arithmetic: an intermediate value rounded too early, a divisor applied twice, a sensitivity coefficient omitted. A deterministic budget carries every intermediate value at full precision and rounds once, by a stated rule. The GUM allows that it may sometimes be appropriate to round uncertainties up rather than to the nearest digit (7.2.6). A calculator that adopts upward rounding of U as a fixed, conservative rule can guarantee that the reported uncertainty is never smaller than the one computed.

5.3 Effective degrees of freedom and the coverage factor

When the combined uncertainty is dominated by components with few degrees of freedom, the coverage factor is taken from the t-distribution at the effective degrees of freedom given by the Welch-Satterthwaite formula (GUM G.4.1):

$$\nu_{\text{eff}} = \frac{u_c^4(y)}{\sum_{i=1}^N \frac{u_i^4(y)}{\nu_i}}$$

The GUM permits either interpolation in Table G.2 or truncation of ν_{eff} to the next lower integer before reading $k = t_p(\nu_{\text{eff}})$; the H.1 example, and the calculator described here, truncate. Truncation is where floating-point arithmetic does its quiet damage. Two equal contributions of 0.07 with 2 degrees of freedom each give an exact ν_{eff} of 4. A straightforward double-precision evaluation of the formula returns 3.999 999 999 999 6, which truncates to 3: the 95 % coverage factor becomes 3.18 instead of 2.78, and U is inflated by 14 %. Held as an exact fraction, ν_{eff} is 4 and the truncation is decided by integer division. Nothing on the card would show the error.

5.4 Accuracy ratios: TAR and TUR

The test accuracy ratio (TAR) compares the tolerance of the unit under test with the accuracy of the standard. ANSI/NCSL Z540.3-2006 defined the test uncertainty ratio (TUR) as the tolerance span divided by twice the 95 % expanded uncertainty of the calibration process. The standard was withdrawn as an active standard in October 2020, but its definition and 4:1 criterion remain widely used. ILAC-G8 defines TUR as the ratio of the tolerance, TL , to the 95 % expanded uncertainty. When TL is interpreted as the

half-width of a symmetric tolerance, this equals the Z540.3 value, but when TL is interpreted as the full span, it yields twice the Z540.3 ratio, so the convention should always be stated. In the Z540.3 form:

$$\text{TUR} = \frac{T_U - T_L}{2 \cdot U_{95}}$$

The decision that matters is whether TUR reaches 4:1. Consider a tolerance of ± 0.010 mm and a calibration process with $U_{95} = 0.002501$ mm. The exact ratio is $0.020 / 0.005002 = 3.9984$. Rounded to the nearest hundredth, it displays as 4.00, which reads as meeting 4:1 when it does not. A deterministic calculator truncates ratios, so it displays 3.99. The rule is a design decision, but only deterministic code applies it every time.

5.5 Decision rules and guard bands

ISO/IEC 17025:2017 requires a laboratory that issues a statement of conformity to state the decision rule it applied. ILAC-G8 describes the common rules. Under simple acceptance, a result is accepted if the measured value lies within the tolerance limits. Under guarded acceptance, the acceptance limits are moved inside the tolerance limits by a guard band w , often a multiple of the expanded uncertainty ($w = r \cdot U$), so that a result is accepted only when the measured value lies inside the narrower acceptance zone. JCGM 106 defines the risks these rules manage as specific and global consumer and producer risks; calibration practice in the Z540.3 tradition calls the global forms the probability of false acceptance (PFA) and of false rejection (PFR).

For a single measured value y with standard uncertainty u and a normal distribution for the measurand, the specific risk that the true value lies outside the tolerance limits T_L and T_U is:

$$P_{\text{out}} = \Phi\left(-\frac{T_L - y}{u}\right) + 1 - \Phi\left(\frac{T_U - y}{u}\right)$$

Consider reading B in Figure 2: a tolerance of ± 0.010 mm, a measured deviation $y = +0.008$ mm, and $u = 0.002$ mm. The reading is within tolerance, yet the specific risk that the item is actually out of tolerance is 15.9%. This is a conditional risk for a single reading, not directly comparable with Z540.3's 2% limit on the global false-accept probability, but it shows why a reading this close to a limit is not safely accepted. A guarded-acceptance rule with $w = U = 0.004$ mm sets the upper acceptance limit at $+0.006$ mm and does not accept it.

These are the calculations where a probabilistic answer does the most harm. The verdict is binary, the boundary is sharp, and a reading close to a limit is exactly the case a technician most needs to get right. Figure 2 shows the zones. Decision zones under simple and guarded acceptance · 3 readings

A guard band withholds acceptance where the risk is highest

Tolerance ± 0.010 mm, $U = 0.004$ mm ($k = 2$), guarded acceptance with $w = U$

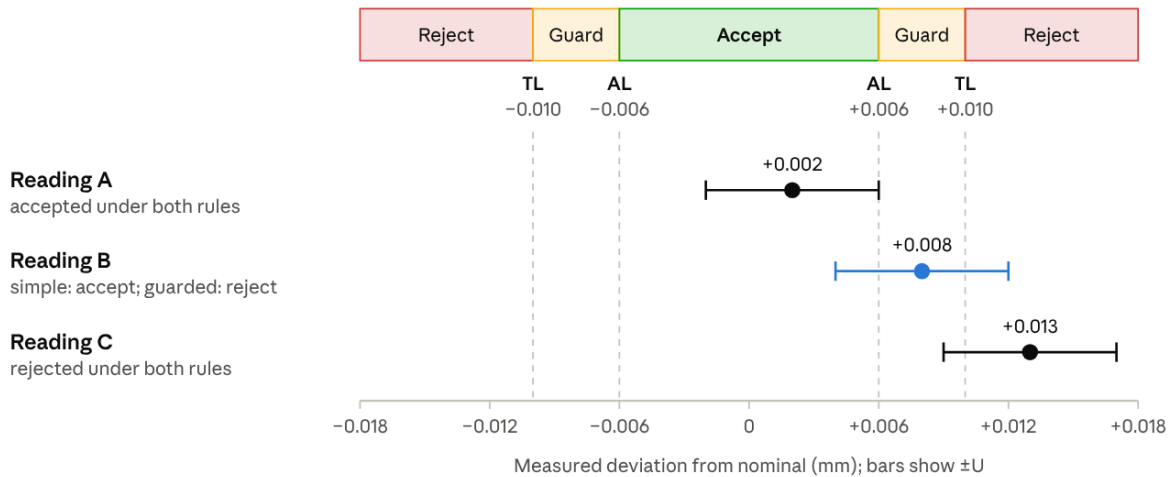


Figure 2. TL = tolerance limit, AL = acceptance limit. Reading B lies inside tolerance, so simple acceptance accepts it, but its uncertainty interval crosses the upper tolerance limit; guarded acceptance does not accept it.

6. Worked example: the GUM gauge-block budget

A deterministic implementation reproduces the published final results of the gauge-block example in JCGM 100:2008 H.1: $v_{\text{eff}} = 16$, $k = 2.92$ and $U = 93$ nm at a 99 % level of confidence. This section follows that one budget through the five stages of Figure 1, as it runs in the MetTutor® Welch-Satterthwaite calculator.

Stage 1, the form. The user enters the uncertainty unit (nm), the level of confidence (99 %) and four contributions:

Contributor	u_i (nm)	Evaluation	v_i
a	25.0	Type B	18
b	9.70	Type A	25.6
c	2.90	Type B	50
d	16.6	Type B	2

Stage 2, the parser. Each value is read as an exact decimal (25.0 becomes 250/10, 9.70 becomes 970/100). The unit, the level, and every degree of freedom are checked. Nothing needs to be refused.

Stage 3, the calculation. Every step is exact until the final display, relative to the input values and the published three-figure table:

- Sum of squares: $625 + 94.09 + 8.41 + 275.56 = 1003.06 \text{ nm}^2$, so $u_c = \sqrt{1003.06} = 31.671\dots \text{ nm}$. (The GUM prints $u_c^2 = 1\,002 \text{ nm}^2$ because it works from unrounded components; the calculator works from the rounded values printed in Table H.1, as a user would enter them.)
- Effective degrees of freedom, formed as an exact fraction:

$$\nu_{\text{eff}} = \frac{1003.06^2}{\frac{625^2}{18} + \frac{94.09^2}{25.6} + \frac{8.41^2}{50} + \frac{275.56^2}{2}} = 16.764\dots \rightarrow 16$$

- Truncated to 16 by integer division, as GUM G.4.1 permits. Table G.2 at $v = 16$ and $p = 99\%$ gives $k = 2.92$ (the unrounded quantile is 2.9208).
- $U = k \cdot u_c = 2.92 \times 31.671\dots = 92.48\dots \text{ nm}$, decided exactly by comparing $k^2 u_c^2$ with the squares of candidate display values.

Stage 4, the presentation. U is rounded up to two significant figures: **93 nm**. u_c is rounded up to three: 31.7 nm. The headline reads " $\nu_{\text{eff}} = 16$, $k = 2.92$ at 99% — $U = 93 \text{ nm}$ ", and the card, the PDF and the Word export carry the same strings.

Stage 5, the explanation. The model receives these values in the computed block. It explains why ν_{eff} is truncated rather than rounded, why component d dominates the denominator despite being smaller than a , and what a 99 % coverage factor means. It quotes 93 nm and does not recompute it.

The GUM reaches 93 nm by multiplying a rounded u_c of 32 nm by 2.92 (93.4 nm). The deterministic calculator reaches the same 93 nm by a different route: it rounds the exact 92.48 nm upward. Both are defensible, and H.1.7 itself notes that including second-order terms would raise u_c to 34 nm. What matters is that the calculator's route is fixed, documented and the same on every run. When the same budget is handed to a language model to compute, there is no guarantee that any of the intermediate values, the truncation or the table lookup is done the same way twice.

7. Validating deterministic calculators

Determinism is what makes validation possible: because each input has exactly one correct output, a calculator can be tested against an independent implementation case by case, and a single mismatch is a finding. The NPL's best-practice guide on software in measurement systems states the underlying concern plainly: the hidden complexity within software is a potential source of undetected errors, and the key issue is to preserve the integrity of the measurement process (Wichmann et al., 2007).

7.1 Methods

The following methods, applied together, give a complete software-validation record of the kind ISO/IEC 17025:2017 7.11 asks for:

Method	What it establishes
Independent reference implementation	A second implementation, written from the standards rather than from the production code, in exact rational arithmetic and sharing no code with production
Random differential testing	Thousands of generated inputs compared value by value with the reference; any mismatch on a displayed value fails
Known answers	Published examples reproduced to their printed precision (GUM, UKAS M3003, Z540.3 Handbook, NIST SP 811)
Edge and rejection cases	Values on every boundary and inputs that must be refused
Invariants	Properties that must hold for any input, such as scaling, reordering and no displayed value contradicting the verdict
Regression suite	A permanent test harness run before every change
Before/after comparison	Every earlier validation input run on the code before and after a change and compared field by field
Live-application tests	Cases entered in the production user interface and exported, with card, export and explanation checked against expected values

Two points of discipline matter as much as the methods. First, the people or processes that build the calculators should not be the ones that verify them. Second, any change to the code, display rules, or prompt text should trigger a new, scoped revision of the validation record before the calculator is relied on again.

7.2 Case study: the MetTutor® deterministic calculators

MetTutor® (mettutor.ai) is an AI-assisted metrology education platform. In 2026, its calculation capabilities were rebuilt using the architecture of Section 4, and eight calculators were validated using the methods above. The roles were as follows. Implementation was carried out by an AI coding assistant working in the production repository. Independent verification, including the specifications, the reference implementations, and the validation records, was carried out by a separate AI assistant session that worked from the standards and the code changes, not from the implementer's results. Live testing, authorization of every change, and approval of every record were carried out by the development team. Because implementation and verification used models from the same family, common-mode error is a real threat to independence; writing the references from the standards, reproducing published known answers, and human live testing mitigate it, but do not eliminate it.

Each reference implementation is written in Python using exact rational arithmetic, with its own rounding rules. Random inputs are generated from fixed seeds across each calculator's accepted input ranges, with cases concentrated near thresholds, table rows, and acceptance limits. A "displayed value" is every number shown on the card or in an export; exact fields must match character for character, and floating-point fields such as risk integrals must agree within a stated relative tolerance.

Calculator	Independent cases	Mismatches	Edge and rejection cases	Live tests
TUR (Simplified)	12,000	0	11	6
TAR	12,000	0	6, plus 9 known answers	6
SI Unit Converter	12,000, plus 177 factors	0	5	6
Type B Estimator	10,000	0	7, plus 1,054 random refusals	6
GUM Uncertainty Budget	8,000 budgets	0	7	5
TUR (Full)	2,000	0	7	4
Decision Rules and Guard Band	6,000	0	35, plus 17 known answers	13
Welch-Satterthwaite	6,000 budgets	0	17, plus 5 known answers	8

Across **68,000 independent random cases**, there were **no mismatches** in any displayed value. **Zero failures** in n cases bounds the per-case mismatch rate at about $3/n$ with **95% confidence**: roughly 0.004% overall and 0.15% for the smallest set of 2,000. **The permanent regression suite passes 3,945 of 3,945 tests at the validated release.** When a later cleanup changed what the calculators display, every earlier validation input (**69,667 in total, including the random cases, edge cases, known answers, and the inputs of the earlier records**) was re-run before and after the change and compared field by field; only the values the change was meant to move had changed.

The validation also uncovered real defects, which is the point of doing it. One was a rule that treated any entered degrees of freedom of 1000 or more as infinite. It changed the coverage factor and the expanded uncertainty in 30 of 6,000 random budgets and was removed. A second was a citation in a calculator's finding that attributed to GUM G.4 a statement the GUM does not make. A third was internal status codes appearing on the user's screen. Each was traced, corrected, and re-validated, and each correction was confined to what the finding identified.

The case study's limits are stated in its records. The validation covers every number on the cards and in their exports. It does not cover the explanatory text the language model writes, which is constrained by prompt rules and checked in live tests but cannot be validated in the same sense (Section 8).

8. Benefits and limitations

The benefits of deterministic computation follow directly from the requirements in Section 3; its limitations are mostly about what it leaves to the language model.

8.1 Benefits

Benefit	What it gives the laboratory, the instructor or the learner
Reproducibility	The same inputs give the same card on every run and every device, so a result can be checked by anyone at any time
Validatability	Each input has one correct output, so the software can be validated case by case under ISO/IEC 17025:2017 7.11
Auditability	Every reported value traces through a documented algorithm to the inputs; an assessor can recompute it
Exactness at boundaries	Truncation, directed rounding and threshold comparisons are decided exactly, so a value is never pushed across 4:1, a table row or an acceptance limit by rounding error
Honest refusal	Inputs that cannot be read unambiguously are refused rather than estimated
Consistency across outputs	One presentation function feeds the card and every export, so they cannot disagree
Better teaching	The model's explanation is anchored to correct computed numbers, which removes the most common source of wrong worked examples
Change control	A regression suite and before/after comparisons show exactly what a change moved

The last benefit is easy to underrate. A language model's arithmetic can change silently when its provider updates the model. **A deterministic calculator changes only when its code changes, and every change can be tested before it is released.**

8.2 Limitations

- **The explanation is still probabilistic.** Prompt rules can tell the model to copy numbers verbatim and to use plain words for internal codes, but an instruction to a language model cannot guarantee every run. In the case study, a later check found internal status codes in the model-written explanation even though the card showed words; a prompt revision corrected it, and a code-level backstop remains available if it recurs. The explanation can be tested by sampling, not validated.

- **Determinism is not correctness.** A deterministic calculator that implements the wrong formula is wrong every time. Determinism makes errors reproducible and therefore findable; independent references, known answers, and review are what find them. The case study's own findings (Section 7.2) were all deterministic defects.
- **Scope must be bounded.** A deterministic calculator handles the models it was built for. Correlated inputs, one-sided or asymmetric specifications, and non-linear models that need Monte Carlo propagation (JCGM 101) each require their own implementation and validation. Outside its scope, a calculator must refuse.
- **Floating-point arithmetic remains where values are irrational.** Risk integrals, π -based factors, and exponentials cannot be exact. IEEE 754 makes basic operations reproducible for a fixed sequence of operations, but library functions such as exp and erf can differ in the last bit across platforms. These values should be computed in a controlled environment and validated against high-precision references using a stated tolerance.
- **Inputs remain the user's responsibility.** A calculator computes exactly what it is given. Choosing the measurement model, the contributors, and the distributions is metrology judgment, and no algorithm removes the need for it.
- **Validation does not transfer automatically.** A laboratory that uses a validated training calculator in its own calibration work must still validate that use for its own purpose.

9. Research and policy context

The architecture proposed here is a strict form of a pattern that machine-learning research has converged on: **let the model reason in language and let a program compute.** Risk frameworks from NIST and the OECD, along with metrology's own guidance on software, point in the same direction.

9.1 Language models and exact arithmetic

Dziri et al. (2023) studied transformers on compositional tasks and reported that off-the-shelf ChatGPT and GPT-4 achieved only 55% and 59% accuracy on three-digit-by-three-digit multiplication. They found that errors early in a computation compound in later steps and argued that performance can decay rapidly as task complexity grows. Chen et al. (2023) give the arithmetic of compounding in a footnote: if each addition is correct with 90% probability, the chance that 50 additions are all correct is below 1% ($0.9^{50} \approx 0.5\%$). An uncertainty budget with a dozen contributors, including squares, a square root, a Welch-Satterthwaite ratio, and a table lookup, is exactly this kind of chained computation. Model capability has improved since these studies, but the structural point stands: a generated number carries no guarantee, however often it is right.

9.2 Delegating computation to code

The research response has been to remove arithmetic from the model. Program-aided Language Models (PAL) have the model write a program and offload the solution step to a Python interpreter; Gao et al. (2023) report that, on the GSM8K arithmetic word-problem benchmark, this approach (using Codex) surpassed PaLM-540B with chain-of-thought prompting by 15 percentage points. Program of Thoughts prompting does the same, explicitly "decoupling complex computation from reasoning and language understanding", with an average gain of about 12% over chain-of-thought across its datasets (Chen et

al., 2023). Toolformer showed that a model can learn to call a calculator and other tools, noting that language models paradoxically struggle with basic functionality such as arithmetic, where much simpler and smaller models excel (Schick et al., 2023).

These approaches still let the model decide what to compute and write the program. For a regulated calculation, that is not enough: the program itself must be fixed and validated. The architecture in Section 4 removes the model from the computation entirely. The model never writes, chooses, or runs the calculation; it receives the validated result.

9.3 Determinism of the computing platform

IEEE 754-2019 specifies that numerical results are uniquely determined by the input data values, the sequence of operations, and the destination formats. That is a guarantee of determinism, but a conditional one: changing the order of operations can change the result. He et al. (2025) showed that this is precisely why LLM inference services are not reproducible in practice, because server load changes the batch size and, with it, the order of floating-point reductions. **A deterministic calculator fixes the order of operations and, where possible, removes the issue altogether with exact rational arithmetic.**

9.4 AI risk frameworks

The NIST AI Risk Management Framework lists the characteristics of trustworthy AI as valid and reliable, safe, secure and resilient, accountable and transparent, explainable and interpretable, privacy-enhanced, and fair with harmful bias managed. It places "**valid and reliable**" at the base as a necessary condition for the others and adopts the ISO 9000 definition of validation: confirmation, through objective evidence, that the requirements for a specific intended use have been fulfilled (NIST AI 100-1, 2023). That definition closely matches the sense in which metrologists validate software under ISO/IEC 17025 (compare VIM 2.45). NIST's Generative AI Profile adds confabulation as a risk specific to generative models (NIST AI 600-1, 2024).

The OECD AI Principles call for AI systems to be robust, secure, and safe throughout their lifecycle (Principle 1.4) and for traceability of datasets, processes, and decisions to enable analysis of a system's outputs (Principle 1.5) (OECD, 2024). Metrology uses "**traceability**" in a narrower sense, but the intent is shared: an output should be explainable by reference to the process that produced it. Deterministic computation is the most direct way to give an AI system's outputs that property.

9.5 Metrology guidance on software

Metrology has long treated calculation software as part of the measurement. The NPL best-practice guide grades validation effort by risk and warns that software's hidden complexity is a source of undetected error (Wichmann et al., 2007). NPL's companion work on reference data sets describes black-box testing against reference data with reference results, the method used in Section 7. NIST's Statistical Reference Datasets provide certified computational results, some computed carrying 500 digits, for testing statistical software (NIST StRD). PTB's TraCIM service extends this to traceability itself, on the principle that all computational links in a traceability chain must be recognized explicitly and known to be operating correctly. **Guidance from accreditation bodies on ISO/IEC 17025:2017 7.11.2**

describes the same practice: compare the system's responses with those expected for a known set of data (IPAC OGC001).

Each of these assumes that the software being tested gives one answer per input. A generative model does not, which is why it belongs on the explanation side of the boundary.

10. Conclusions and recommendations

Language models are valuable in metrology education and practice for what they do well: explaining, teaching, guiding a technician through a procedure, and answering the question behind the question. They should not produce the numbers. **Every requirement that gives a measurement result its authority (traceability, software validation, a defined uncertainty procedure, an agreed decision rule) assumes a calculation that gives one answer per input. Deterministic code meets that assumption; a generative model does not.**

The architecture described here retains both strengths. Code computes every reported value exactly, rejects what it cannot read, and presents the same strings across all surfaces. The model receives those strings and explains them. The case study shows that this can be validated to the standard an assessor would expect: 68,000 independent cases with no mismatches, published examples reproduced, and every change revalidated before release.

Recommendations for developers of AI systems for metrology:

1. Compute every reported number in deterministic code. Give the model finished values and forbid it to recompute, re-round, or dispute them.
2. Use exact rational arithmetic wherever a value can be exact; confine floating-point to irrational quantities and validate those against high-precision references.
3. Refuse ambiguous or out-of-scope input rather than estimate an answer.
4. Make display rules conservative and explicit: round uncertainties up, truncate ratios, never show a value on the wrong side of a limit.
5. Keep instructions and user data in separate messages.
6. Validate each calculator against an independent reference, published known answers and edge cases, and keep the record under revision control.
7. State plainly what is not validated, in particular the model-written explanation.

Recommendations for laboratories, educators and assessors:

1. Ask any AI tool that reports a measurement value or uncertainty where the number was computed and how that computation was validated.
2. Treat a number generated by a language model as unverified until it has been reproduced independently.
3. Apply the existing ISO/IEC 17025 software-validation requirements to AI tools exactly as to any other calculation software. No new principle is needed; apply the existing one consistently.

Determinism does not make an AI system intelligent. It makes the system's numbers trustworthy, which in measurement science is the precondition for everything else.

References

Metrology standards and guidance

1. JCGM 100:2008. [*Evaluation of measurement data — Guide to the expression of uncertainty in measurement*](#) (GUM). Joint Committee for Guides in Metrology, BIPM. DOI 10.59161/JCGM100-2008E.
2. JCGM 101:2008. [*Evaluation of measurement data — Supplement 1 to the GUM — Propagation of distributions using a Monte Carlo method*](#). JCGM, BIPM.
3. JCGM 106:2012 / OIML G 1-106:2012. [*Evaluation of measurement data — The role of measurement uncertainty in conformity assessment*](#). JCGM.
4. JCGM 200:2012. [*International vocabulary of metrology — Basic and general concepts and associated terms \(VIM\)*](#), 3rd ed., definition 2.41. JCGM, BIPM.
5. ISO/IEC 17025:2017. [*General requirements for the competence of testing and calibration laboratories*](#). ISO/IEC.
6. ILAC-G8:09/2019. [*Guidelines on Decision Rules and Statements of Conformity*](#). ILAC. Reissued without substantive change by the Global Accreditation Cooperation as Global ACI-TECH-1-002 (G).
7. ANSI/NCSL Z540.3-2006 (R2013). *Requirements for the Calibration of Measuring and Test Equipment*. NCSL International. [Withdrawn as an active standard in October 2020](#); its requirements remain in use through supplemental accreditation programs such as [ANAB SR 2402](#).
8. NCSL International (2009). [*Handbook for the Application of ANSI/NCSL Z540.3-2006*](#).
9. NCSL International (2014). [*RP-18: Estimation and Evaluation of Measurement Decision Risk*](#).
10. A2LA (2022). [*G136: Guidance on Decision Rules in Calibration*](#). American Association for Laboratory Accreditation.
11. Taylor, B. N., and Kuyatt, C. E. (1994). [*Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results*](#). NIST Technical Note 1297.
12. Newell, D. B., and Tiesinga, E. (eds.) (2019). [*The International System of Units \(SI\)*](#). NIST SP 330.
13. CGPM (2018). [*Resolution 1 of the 26th CGPM: On the revision of the International System of Units*](#). BIPM.
14. Thompson, A., and Taylor, B. N. (2008). [*Guide for the Use of the International System of Units \(SI\)*](#). NIST SP 811.
15. National Bureau of Standards (1959). [*Refinement of Values for the Yard and the Pound*](#). Federal Register 24 FR 5348.
16. ASQ. [Certified Calibration Technician \(CCT\) Body of Knowledge](#).

Software validation in measurement

17. Wichmann, B., Parkin, G., and Barker, R. (2007). [*Validation of Software in Measurement Systems*](#). Software Support for Metrology Best Practice Guide No. 1, NPL Report DEM-ES 014, v2.2. National Physical Laboratory.
18. National Physical Laboratory. [Report DEM-ES 4 on reference data sets for testing metrology software](#).

19. NIST. [Statistical Reference Datasets \(StRD\)](#), Standard Reference Database 140. DOI 10.18434/T43G6C.
20. PTB. [TraCIM: Traceability for computationally-intensive metrology](#). Physikalisch-Technische Bundesanstalt.
21. IPAC (2018). [OGC001: Guide for the application of NP EN ISO/IEC 17025](#). Instituto Português de Acreditação.
22. IEEE (2019). [IEEE Std 754-2019, IEEE Standard for Floating-Point Arithmetic](#).
23. Goldberg, D. (1991). [What Every Computer Scientist Should Know About Floating-Point Arithmetic](#). *ACM Computing Surveys* 23(1). DOI 10.1145/103162.103163.

Artificial intelligence: research, risk and policy

24. NIST (2023). [Artificial Intelligence Risk Management Framework \(AI RMF 1.0\)](#). NIST AI 100-1.
25. NIST (2024). [Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile](#). NIST AI 600-1.
26. NIST. [AI Resources](#).
27. OECD (2019, updated 2024). [Recommendation of the Council on Artificial Intelligence](#) (OECD AI Principles), OECD/LEGAL/0449. See also [OECD: Artificial intelligence](#).
28. Dziri, N., Lu, X., Sclar, M., et al. (2023). [Faith and Fate: Limits of Transformers on Compositionality](#). NeurIPS 2023. arXiv:2305.18654.
29. Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. (2023). [PAL: Program-aided Language Models](#). arXiv:2211.10435.
30. Chen, W., Ma, X., Wang, X., and Cohen, W. W. (2023). [Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks](#). arXiv:2211.12588.
31. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). [Toolformer: Language Models Can Teach Themselves to Use Tools](#). arXiv:2302.04761.
32. He, H., and Thinking Machines Lab (2025). [Defeating Nondeterminism in LLM Inference](#). Thinking Machines Lab.
33. Wikipedia. [Deterministic algorithm](#).

Case study records

34. Met Scientia, LLC (2026). *MT-VAL-SUM-001 Rev B: MetTutor® Deterministic Calculators — Validation Summary*, with validation records MT-VAL-TURS-002, TAR-002, SI-002, TB-002, GB-002 and TURF-002 (Rev B) and MT-VAL-DR-002 and WS-001 (Rev E).

About the Metrology Institute

The Metrology Institute is a 501(c)(3) nonprofit foundation dedicated to advancing measurement science as a recognized STEM discipline and to strengthening the workforce that depends on it. The Institute partners with educators, industry experts, professional instructors, and accredited educational institutions to expand access to rigorous measurement science education and to establish clearer pathways into metrology, calibration, and quality careers.

From K-12 classrooms to professional practice, the Institute develops and supports technical handbooks, curriculum, training resources, and educational programs grounded in authoritative measurement standards and in the practices of working calibration laboratories and quality systems. These resources help educational institutions introduce measurement science without having to build specialized metrology expertise from the ground up.

The Institute also supports research into the application of artificial intelligence to measurement science education, including technologies that can make technical learning more accessible, adaptive, and scalable. Through programs, partnerships, technical resources, and emerging educational technologies, the Metrology Institute is working to make measurement science more visible in education, valued by industry, and accessible to the next generation of measurement professionals.

Empowering Minds Through Measurement

metrologyinstitute.org