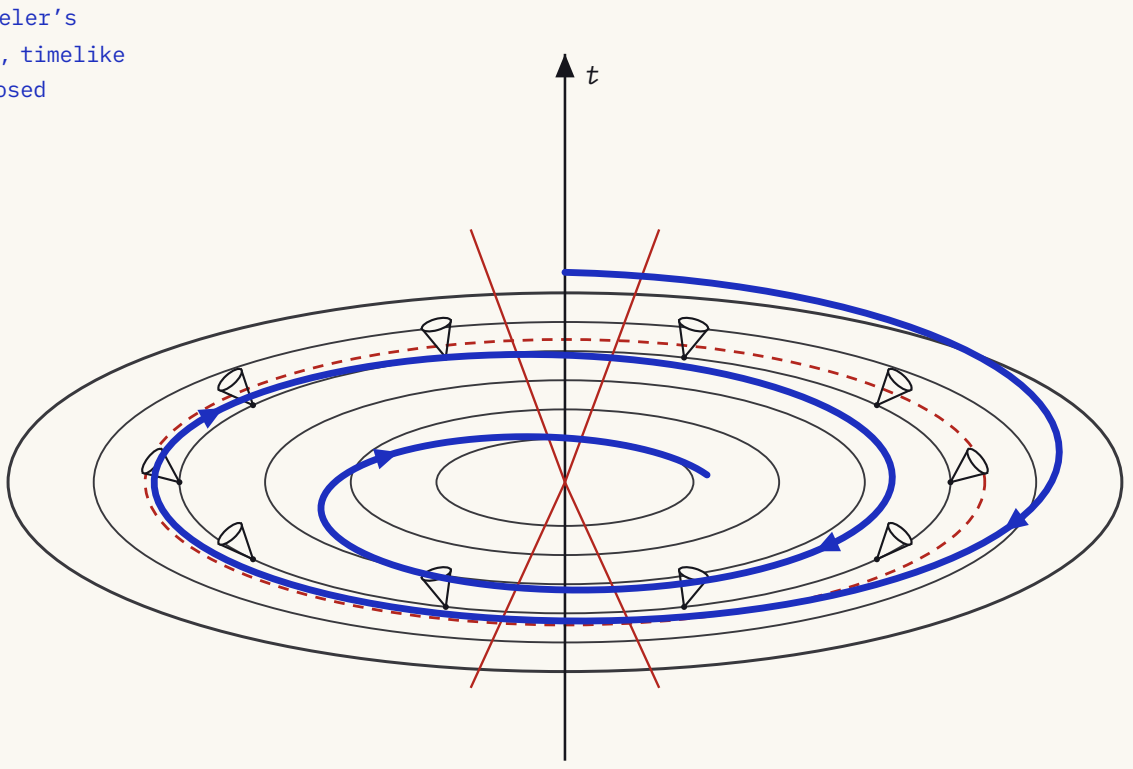
WHITEPAPER

VeriPIE

Verifiable Predictive Intelligence Engine

Operational autoformalization for verifiable decisions



"We had better be quite sure that the purpose put into the machine is the purpose which we really desire…"
Norbert Wiener, 1960

Brian Li, Co-founder & CTO
KNOWIDEA

AUGUST 2026 · v1.9

Abstract

Language models can turn short business questions into detailed plans, but a fluent plan does not show where its numbers came from or whether its formal model preserves the decision a person intended to make. VeriPIE is a closed-source decision system that translates an operational question into an explicit program of levers, limits and measurable outcomes. Untrusted models and solvers may propose that program and a candidate plan. Separate checkers then record the origin of every influential number, compare the program with the stated decision, and recheck feasibility and selected optimality claims in exact rational arithmetic. The system may refuse to issue a verified decision when those obligations do not close.

This whitepaper describes the verification boundary, the role of the decision compiler, and a verifier-guided feedback loop that is implemented during inference and review. It also reports a 20-case internal benchmark derived from de-identified consulting use cases. On the reported process measures, VeriPIE elicited more load-bearing facts, disclosed more assumptions, recorded more origins, and reported sensitivity thresholds more often than the SOTA-LLM baseline. The benchmark measures process discipline rather than business decision quality. Reinforcement learning with verifiable rewards and reinforcement learning via symbolic feedback, including the proposed reward vector, remain future research directions.

VeriPIE at a glance

VeriPIE is a decision system for quantitative operational choices where a person remains accountable for the result. It turns a business question into an explicit program, records where its numbers came from, and issues a certificate only when the required checks and human approval close.

The need A fluent plan can hide unsupported numbers or a model that does not preserve the original decision.	**Where it fits** Leaders and analysts making measurable choices about capital, staffing, capacity, pricing, maintenance, migration or risk.
What the system does Compiles the question into levers, limits and outcomes; records provenance; solves the program; and checks the result.	**What may be verified** Arithmetic, feasibility, selected optimality claims, coefficient ranges and selected translation properties of the declared program.
Current evidence On a 20-case internal benchmark, VeriPIE improved fact elicitation, assumption disclosure, provenance and sensitivity reporting over a SOTA-LLM baseline.	**The boundary** A certificate does not prove that a fitted effect will occur, that every modelling choice reflects intent, or that the recommendation is a good business decision.

How to read the workflow

Language models and solvers may propose the program and candidate plan. The decision compiler, provenance ledger and verification kernels determine which claims may enter the certificate. A named owner still decides whether the declared model is suitable for the real decision. A failed obligation can trigger repair or refusal during inference and review. Training with verifiable rewards remains future work.

Definitions used in this paper

The paper uses several terms in a narrow way. This section gives their plain meaning before the technical details begin.

From a question to a program

Term	Plain meaning
Operational question	A business question that names a measurable result and can be acted on through specific choices.
Decision lever	Something the decision maker can change, such as headcount, budget allocation or the number of maintenance packages.
Outcome	The measurable result the organisation wants to improve, such as revenue, downtime or risk.
Constraint	A limit the plan must respect, such as a budget, capacity cap, minimum service level or deadline.
Operational program	The mathematical version of the decision: its levers, outcomes, constraints and the allowed range of each lever.
Decision compiler	The part of VeriPIE that turns the recorded question into an operational program. Its output is a proposal that still must be checked.
Provenance ledger	A record of where every influential number came from, including supplied facts, external sources, fitted values and assumptions.

From a candidate to a certificate

Term	Plain meaning
Candidate plan	A proposed setting of the decision levers. It is an answer to check, not a verified result.
Solver	Software that searches the operational program for a feasible or high-scoring candidate plan. VeriPIE does not trust the solver on its word.
Verification kernel	A smaller checker that independently tests a specific claim, such as feasibility, arithmetic or a translation property.
Exact rational arithmetic	Calculations performed with fractions instead of rounded floating-point numbers, avoiding that source of numerical error.
Decision certificate	A typed record that binds the program, number origins, checks, sensitivity ranges, overrides, approval and final verdict together.
Sensitivity certificate	A statement of how far selected fitted values may change before the recommended plan changes. It is not proof that the fitted values are correct.
Trust boundary	The line between components allowed to propose results and the checkers or people required to approve those results.
Refusal	A result that returns the work completed so far but withholds a verified recommendation because a required check or modelling condition failed.

Feedback and learning terms

Term	Plain meaning
Verifier-guided decision making	The current feedback loop in which checker findings can cause the system or reviewer to repair, retry or stop a decision during inference and review.
RLVR	Reinforcement learning with verifiable rewards. A checker supplies an objective reward for an outcome that can be tested automatically. This is a future direction, not a capability implemented in the current system.
RLSF	Reinforcement learning via symbolic feedback. Jha and colleagues propose using certificates from symbolic tools to provide fine-grained feedback during training [17]. This is also future work.
Reward vector	A proposed set of separate training signals for parsing, provenance, translation, feasibility, optimality and owner approval. Keeping the signals separate shows which obligation failed.

The central distinction

A certificate can prove claims about the declared program. It cannot prove that the program captures the whole business problem or that a fitted effect will occur in the real world. Those judgements remain visible and require an accountable person.

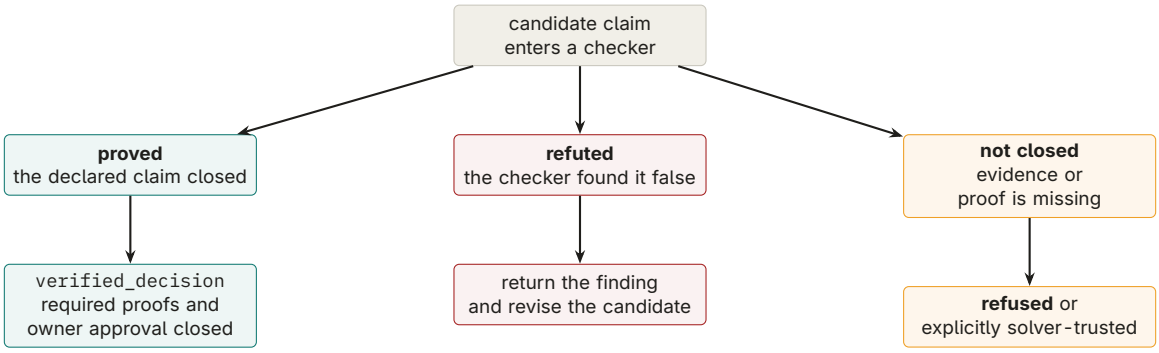


Figure 1: How to read the main verdict words. A mathematical claim and the final decision have different approval conditions.

Fluent plans hide unverifiable numbers

Language models can produce confident operational plans from short prompts. Ask how to grow revenue in a new market, and the answer may include headcount, a channel mix, a budget split and a timeline. It may read as if it came from someone who has done the work before. The difficulty is tracing the plan's numbers to evidence instead of prose added to complete the answer. At delivery, a reader may not be able to tell which figures came from evidence and which were added to complete the prose. The reader then has no sound basis for deciding which parts of the plan are safe to use.

Recent consulting failures provide concrete examples. In 2025, Deloitte Australia agreed to refund part of a government contract after a report included fabricated citations and a false court quotation; the revised report disclosed the use of generative AI [3]. In 2026, EY withdrew a loyalty report after researchers found apparent AI-generated errors and false footnotes [12]. KPMG later withdrew a report on agentic AI after several organisations disputed case studies attributed to them [13]. These cases show why polished consulting prose cannot replace evidence, provenance or review.

Common repairs leave this problem unresolved. A request for citations still asks the model to generate text that must itself be checked. Self-review may repeat the failure mode under review. Adding an optimiser verifies neither the provenance of the numbers nor the translation that produced the model. A solver can optimise the program it receives with great precision while remaining silent about how that program was constructed.

We argue that verification becomes useful after a strategic goal has been translated into an operational program. The value comes from the trust boundary around the solver, not from the solver alone. VeriPIE starts with a goal for which a person is accountable. It translates that goal into a formula, levers and limits, then solves the program. A small kernel may refuse the result unless it can re-derive the quantitative claims with exact rational arithmetic. The program is serialised. A later planning cycle can therefore revise and rerun the same explicit model instead of starting with another unrecorded act of reasoning.

Why VeriPIE, why now, and for whom

Language models are moving from drafting documents into recommending actions. That shift raises the cost of a hidden assumption. An incorrect phrase in a draft can be edited. An unsupported budget, staffing plan or maintenance schedule can move money, people and equipment before anyone notices the weak link in the reasoning. High-stakes organisations therefore need a visible record of the inputs, the model, the proof boundary and the person who approved the decision.

VeriPIE is intended for leaders and analysts who are accountable for measurable operational choices under explicit limits. Typical questions involve capital allocation, workforce plans, capacity, pricing, maintenance, migration, collections or retention. Likely users and buyers sit in operations, finance, strategy, risk and internal consulting teams, including teams in oil and gas, finance, manufacturing and other high-stakes industries.

The system is not a good fit for every question. It cannot turn an open-ended vision into a verified decision when no measurable outcome, decision lever or limit has been defined. It does not establish causality from sparse observations, and it is not designed to make autonomous decisions without an accountable human reviewer. In those settings, the useful result may be a

list of missing facts or a refusal rather than a recommendation.

Models and solvers are replaceable proposal components. VeriPIE's product layer is the decision compiler, provenance ledger, translation checks, certificate workflow and organisation-specific history of corrections and approvals. General-purpose language models can generate plausible plans. VeriPIE is built around the narrower task of deciding which quantitative claims may be certified, which remain assumptions, and when the system must stop.

One worked decision shows the full path

Consider a hypothetical manufacturer choosing preventive maintenance packages and operator training cohorts for the next quarter. The goal is to reduce downtime subject to a budget and delivery limits. All values in this example are illustrative. It is not a client case or a benchmark question.

The decision compiler records two integer levers: maintenance packages m and training cohorts t. The stated budget is 1,000 thousand dollars. A package costs 120 thousand dollars and a cohort costs 70 thousand dollars. The organisation can deliver at most four packages and eight cohorts. A fitted planning model estimates a reduction of 18 downtime hours per package and nine hours per cohort. The resulting program is

Illustrative operational program

$$\begin{aligned} &\textit{maximise} && 18m + 9t \\ &\textit{subject to} && 120m + 70t \le 1000, \\ & && 0 \le m \le 4, \qquad 0 \le t \le 8, \\ & && m, t \in \mathbb{Z}. \end{aligned}$$

The candidate plan is $m = 4$ and $t = 7$. It costs 970 thousand dollars and the model estimates a reduction of 135 downtime hours. The exact kernel enumerates all 45 bounded integer settings, removes the infeasible settings, and confirms that no feasible setting has a larger objective value.

Stage	What the decision record contains
Question	Reduce quarterly downtime through maintenance and training.
Inputs	Budget, unit costs and delivery caps are stated; effect coefficients are fitted from internal cases.
Program	Two bounded integer levers, one budget constraint and one fitted objective.
Candidate	Four maintenance packages and seven training cohorts, costing 970 and yielding 135 modelled hours.
Checks	Provenance, translation, exact feasibility and exact optimality under the serialised program.
Approval	A named decision owner confirms that the levers and fitted effects are suitable for this planning decision.

Table 1: The worked example follows one decision from question to approval.

The sensitivity record makes the fitted assumptions easier to challenge. With the maintenance effect fixed at 18 hours, the selected plan changes from $(4, 7)$ to $(3, 8)$ only when the training

effect exceeds 18 hours per cohort. With the training effect fixed at nine hours, the same alternative becomes better when the maintenance effect falls below nine hours per package. The equality points are ties. These are local, one-coefficient changes under the same feasible set, not forecasts about future downtime.

The final verdict may be *verified decision* only after the program checks close and the accountable owner approves the modelling judgement. The certificate proves the solution of the declared program. It does not prove that the fitted reduction in downtime will occur.

VeriPIE makes operational autoformalization verifiable

This paper makes five contributions.

1. **Verification becomes useful at the operational program.** Strategic advice is too open-ended to verify directly. Verification becomes useful once the advice is expressed through explicit levers, limits and measurable outcomes.
2. **Operational autoformalization requires a translation kernel.** VeriPIE translates a natural-language decision question into a formal optimisation program. The kernel checks selected properties of that translation, and its findings affect the result.
3. **Faithfulness checks must preserve constraint direction.** A constraint can contain the correct variable and value while expressing the wrong feasible set. The translation kernel therefore distinguishes a floor from a ceiling instead of checking only that a limit is present.
4. **Inductive fits require sensitivity certificates rather than proof claims.** A fitted formula cannot be proved correct from the observations used to fit it. The system instead certifies how far selected coefficients may vary before the recommendation changes.
5. **Refusal is a valid system result.** Any stage may stop the run, identify what is missing, and return the work completed so far.

VeriPIE does not claim to produce correct plans. It produces plans whose derivation can be inspected and whose unverifiable steps are labelled. The design records this distinction throughout the workflow.

The paper supports these contributions with the implemented architecture, soundness arguments for arithmetic and sensitivity claims, a reproduced failure analysis, and a 20-case operational benchmark. The benchmark measures elicitation, disclosure, provenance, constraint handling, sensitivity and named unknowns. It does not estimate whether the resulting business recommendations are correct.

Formal correctness does not establish faithful translation

Mathematics, natural science and operational planning face a similar problem. In each field, a machine can propose a formal object faster than a person can inspect it. The object may be a proof, a closed-form law or an optimisation program. Existing tools can check a proof, solve a program or repeat an arithmetic derivation. On their own, they cannot show that the formal object faithfully represents the claim, observations or decision that led to it. The hard question is often about correspondence, not calculation.

In mathematics, the statement is the risk. Autoformalization translates informal mathematics

into a proof assistant such as Lean [19], whose kernel accepts or rejects the proof but has no opinion whatever about the statement: a machine-checked proof of a mis-stated theorem is a perfectly valid proof. The mitigations that field has converged on [27] are the ones we use, for the same reason. Back-translation, where the formal statement is rendered back into language and compared with the original; agreement across independent samples [24]; and human review aimed at the statement rather than the proof. Benchmark suites make this tractable by providing or assuming a target formalisation against which a translation can be compared [4, 28]. Operational planning has no equivalent reference program.

In science, a good fit does not establish the law. Symbolic regression [22] recovers closed-form laws from observations, and its difficulty is the same one relocated. An expression that fits the data need not be the law generating it; coefficients are unidentifiable when the data does not vary in the right directions; an exactly determined fit interpolates and establishes nothing. The response in that literature is to constrain what may be proposed, dimensional analysis and symmetry being the familiar examples [23]: where a formal object cannot be validated after the fact, restrict the proposals.

In decisions, the program determines what the optimum means. The third instance is operational planning. **We treat the translation of an operational question into an optimisation program as a form of autoformalization.** In mathematics, the input is an informal theorem and the output is a formal statement in a language such as Lean. Here, the input is a natural-language decision question and the output is a formal operational program: variables, bounds, coefficients, constraints and an objective. A Lean kernel can check a proof of the statement it receives, while a solver can check an optimum of the program it receives. Neither establishes that the formal object preserves the meaning of the natural-language input.

This operational version is harder in two respects. An operational goal is genuinely underdetermined: there is no unique correct formalisation of more business in Houston and no reference to score against, so faithfulness has to be monitored while the run proceeds rather than established once. The numbers are also incomplete. A theorem arrives with its hypotheses, while an operational problem arrives with many coefficients missing, which is why so much of what follows concerns provenance.

The three fields suggest a common discipline. Capable but untrusted machinery may propose a formal object, while a smaller trusted component checks the properties that admit exact verification. Steps that cannot be verified must be identified separately and bounded where possible. VeriPIE applies this discipline to operational planning, where an error can pass directly from a generated model into a decision.

The cover illustrates the same separation between validity within a model and correspondence with the world. Gödel's rotating universe [15] satisfies Einstein's field equations but does not describe the observed universe. Its light cones tip over past a critical radius until a closed circle in space becomes a curve in time. The consequence follows from the model; the formalism alone cannot establish that the model describes our world.

Verification becomes useful at the operational program

A user may begin with a goal such as more business in Houston. At that level, the request admits many reasonable interpretations and offers no single object that can be verified. The system therefore asks how the goal could be acted on. Repeating that question eventually produces levers with ranges, a measurable outcome, and limits within which the levers compete. These

elements define an operational program. They also mark the point at which numerical precision becomes useful, because each coefficient now affects a specific feasible decision.

An interview turns the goal into this program. Two rules govern the interview. First, each question must change the model. A question that changes no variable, bound or coefficient wastes the time and attention of the person answering. Second, an estimate must never pass as a measurement. If a response coefficient is a guess, the system records it as a guess with a plausible range. Every later stage depends on that record. The endpoint can now be stated precisely.

Definition 1 (An operational program makes the decision explicit)

An operational program is a vector of levers with ranges. By default, the outcome is affine. If the interview explicitly states non-increasing response tranches, the compiler expands a lever into band variables. Let $x = (x_1, \ldots, x_n)$ denote this expanded vector, with $\ell \le x \le u$. The compiled outcome is

$$f(x) \;=\; \alpha + c^{\top} x,$$

and limits $Ax \le b$ that make the levers compete. A plan is a setting of the levers, and the recommended plan is a maximiser of f over the feasible set $X = \{\, x : Ax \le b,\ \ell \le x \le u \,\}$.

When a goal has several outcome axes, the system combines affine outcomes $f_1, \ldots, f_k$ using a declared, versioned weight vector w, $f_w(x) = \sum_{j=1}^{k} w_j f_j(x)$. Any optimality certificate is conditional on that vector. The system separately checks the weight-free Pareto claim: there is no feasible plan that is at least as good as the selected plan on every outcome axis and strictly better on at least one.

Every quantity in this object is a number somebody has to answer for. The rest of the system sources or derives what the interview did not supply, and labels any remaining value as an assumption. Levers that are whole by nature keep their integrality, which is where a certificate can be left open.

Untrusted systems propose and trusted kernels decide

The architecture separates proposals from verification. A language model, retrieval system and optimisation solvers may propose inputs and results. The arithmetic kernel uses exact rational arithmetic and accepts a serialised program made of plain numbers, strings, lists and dictionaries. No solver object crosses the boundary. It repeats the feasibility and value calculations instead of importing them from a solver. A separate translation checker compares the compiled program with the recorded interview.

These checking cores are not the entire trusted computing base. End-to-end trust also depends on schema validation, the offline decision-certificate checker, deterministic serialisation, the interpreter and the runtime. Generated prose, retrieval results and solver internals remain outside the trusted base. This paper therefore claims architectural separation and checkable certificate artifacts, not public source auditability or formal verification of the runtime and report renderer.

Every solver and search method is treated as untrusted. Z3 [10] is a satisfiability modulo theories solver [5] from Microsoft Research; it received the 2015 ACM SIGPLAN Programming

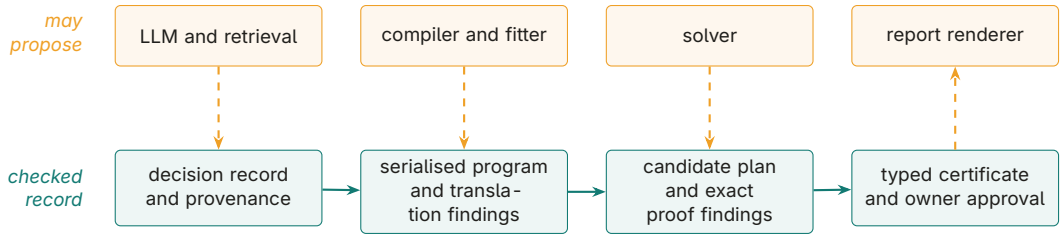


Figure 2: The implementation as a flow of checked artifacts. Proposal and presentation components may change without becoming part of the proof. The certificate binds the recorded decision, serialised program, checked candidate and accountable approval.

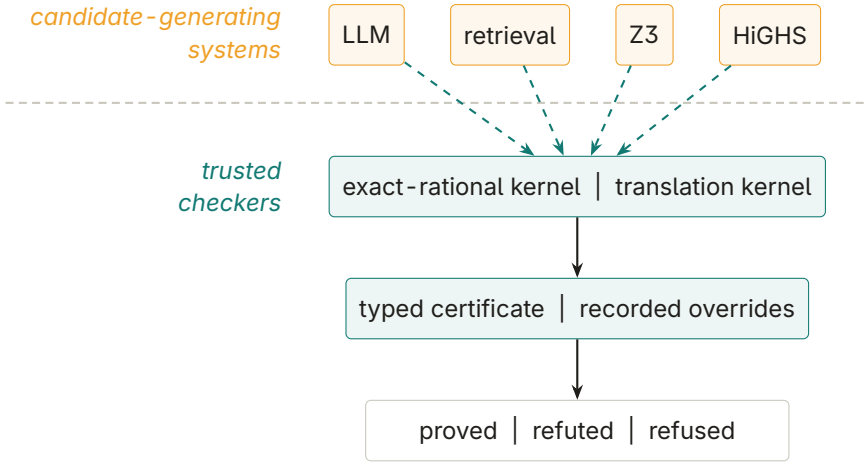


Figure 3: The trust boundary. Systems above the line generate candidate inputs and results. Kernels below the line independently check the claims required for the final verdict.

Languages Software Award, and at Microsoft alone it has discharged more than a billion constraint queries for the whitebox fuzzing that hunts security vulnerabilities in shipped software. On finite problems it is asked to find a feasible sequence with a better objective value than the current candidate. Any sequence it finds is checked by the kernel. If it reports that no better sequence exists, the kernel confirms that result by exact enumeration when the set is small enough. Otherwise the result is labelled solver-trusted, meaning that the verdict relies on the solver rather than a complete kernel proof. HiGHS [16] is the open-source linear and mixed integer solver built on the parallel dual revised simplex work at Edinburgh, and the engine SciPy adopted for linear programming. It is the primary solver in this system and proposes the lever setting that maximises the fitted formula within the limits. The Bellman recursion [6] is the oldest of the three and needs no solver at all, which makes it the one method whose entire search the kernel re-runs outright, in exact rationals, for staged decision processes. All three follow the same rule: the system reports a proposed result as verified only after the kernel has re-derived the relevant claim.

Exact rechecking prevents numerical error from creating a false proof. HiGHS proposes a setting in floating point. The setting is rationalised, re-checked for feasibility, and bounded from above by a dual vector, all in exact arithmetic.

Before certification, the bounded program is put into the nonnegative form used below. The checker substitutes $z = x - \ell$, appends the upper bounds $z \le u - \ell$ to the constraint matrix, and rewrites the remaining limits as $Az \le b - A\ell$. The constant $\alpha + c^{\top}\ell$ does not affect which plan is optimal and is added back when the objective value is reported.

Certificate field	Illustrative value	What it establishes
Program identifier	`VP-EXAMPLE`	Binds the verdict to one serialised program.
Model review	confirmed	A decision owner accepted the represented model.
Provenance ledger	complete	Every influential number has a typed origin.
Translation check	no blocking finding	Published faithfulness obligations closed.
Feasibility	proved	The selected setting satisfies the recorded limits.
Optimality	proved under w	The selected setting is optimal for the declared weights.
Sensitivity	ranges attached	The report states where the recommendation remains stable.
Overrides	none	No blocking finding was accepted by exception.
Decision verdict	`verified_decision`	Model approval and required solver proof both closed.

Table 2: Illustrative decision-certificate structure. The values show field semantics only; they are not a benchmark result or a client recommendation.

Lemma 2 (An exact upper bound certifies optimality)

For a normalised maximisation program with feasible set $\{z : \widetilde{A}z \le \widetilde{b},\ z \ge 0\}$ and objective c, any $y \ge 0$ with $\widetilde{A}^\top y \ge c$ certifies $c^\top z \le y^\top \widetilde{b}$ for every feasible z, whether or not y is optimal. If moreover $c^\top z^\star = y^\top \widetilde{b}$ holds in exact arithmetic, then $z^\star$ is optimal and y is an optimal solution of the dual.

The proof is the whole of weak duality, $c^\top z \le y^\top \widetilde{A} z \le y^\top \widetilde{b}$, with both conditions on y checked in exact rationals. Numerical error inside HiGHS may produce an upper bound that is too loose to certify the candidate, but it cannot create a false proof. When the exact upper bound equals the candidate's exact objective value, the candidate is optimal. The dual entries may then be reported as certified marginal values, which state how much one additional unit of each limit is worth. When an integer problem leaves a difference between the candidate value and the certified bound, the verdict is solver-trusted rather than proved.

A decision certificate separates evidence, proof and approval

A decision certificate is the hand-off between the product and the decision owner. It does not turn assumptions or approval into mathematical proof. It records them beside the claims the trusted checkers can establish. The final verdict is issued only when the model review is confirmed and the required solver claims are proved. Table 2 shows the public structure of that record without exposing a client case or proprietary rule inventory.

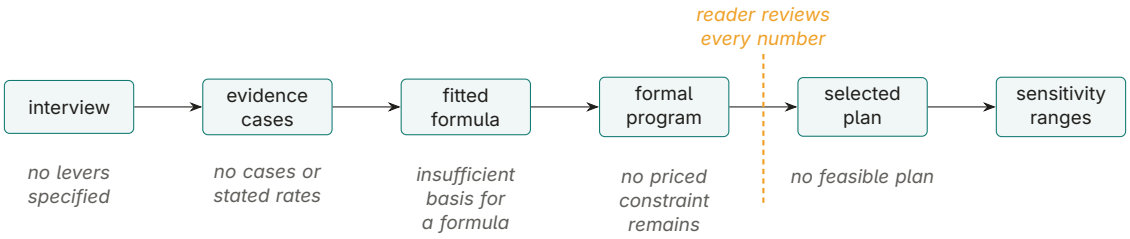


Figure 4: The six stages and the condition that stops the run at each stage. Before solving, the reader may review every number in the formal program.

Operational questions must be compiled before they can be verified

A run has six stages: interview, cases, formula, program, plan and ranges (Figure 4). Any stage may stop the run. When that happens, the system identifies the missing requirement and returns the completed stages. This behaviour prevents output requirements from forcing unsupported results. For example, a fitted formula with more coefficients than the cases can identify may still look authoritative unless the fitting stage refuses it.

Comparable cases determine what can be fitted. The formula is induced from comparable cases, so an invented row can change a coefficient and then change the recommendation. The reader's own history is considered first. Structured sources that attach citations to individual fields come next, followed by plain web search. A case missing any lever value is dropped instead of being completed by inference. A cited URL that was not retrieved is treated as fabricated. The reader's results are not pooled with those of other organisations unless the reader's cases cannot support the fit alone. When pooling is necessary, the report identifies it because it assumes that another organisation's response rates transfer to the reader.

The fitted formula is affine by design. Given m cases, each a lever setting $x^{(i)}$ with its outcome $y^{(i)}$, the coefficients are the least-squares solution of the normal equations

$$(X^\top X)\,\hat{\beta} \;=\; X^\top y, \qquad X \;=\; \begin{pmatrix} 1 & x^{(1)\top} \\ \vdots & \vdots \\ 1 & x^{(m)\top} \end{pmatrix},$$

solved by Gauss-Jordan elimination over exact rationals, so the arithmetic contributes no error of its own. The fitting stage refuses a result when X fails the rank conditions described in A fitted formula cannot be verified. An exactly determined fit passes through every point, which is interpolation rather than evidence, and is labelled as such. A fit explaining less than half the variation between cases is flagged as evidence that an important lever may be missing. When no fit is possible, the run uses rates stated in the interview, labels any remaining values as assumptions, and records an origin of stated, assumed or mixed as data rather than prose. The fitted structure itself is never searched: it is fixed affine and only the coefficients are induced. The compiled program may also contain non-increasing response tranches, but only when the interview explicitly states them. Each tranche becomes a band variable, so the expanded program remains linear and the dual certificate still applies. The repair loop removes tranches invented by the compiler. This restriction preserves the convex optimality region in A fitted formula cannot be verified and keeps its intervals certifiable.

A limit becomes a constraint only when its cost is represented. A forty thousand dollar travel

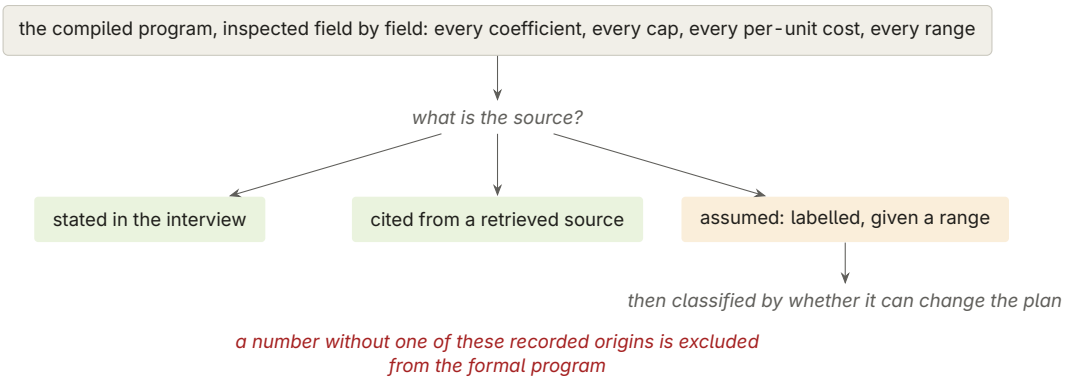


Figure 5: The ledger is built by inspecting every quantitative field in the program that will be solved. Green entries trace to a person or source. Amber entries are declared assumptions that are later tested for their effect on the recommended plan.

budget does not constrain the program until the cost of a trip is represented. Missing per-unit figures are derived from published components where possible; otherwise they are assumed and labelled. Two safeguards came from observed failures. A second compilation after adding an assumed cost is retained only if it adds constraints, since a new model response may omit existing structure. If no constraint remains after compilation, the run stops because setting every beneficial lever to its maximum does not represent the intended resource allocation problem.

The ledger records every number before it is used. Before solving, the ledger inspects the compiled program field by field and records the origin of every coefficient, cap, per-unit cost and range. A figure that traces to the interview or a retrievable source is recorded as stated; anything else is recorded as an assumption. Completeness follows from inspecting the program that will be solved instead of relying on each earlier code path to describe its own output. A number either appears in the ledger with its provenance or cannot enter the program (Figure 5). The run may pause at this point so the reader can review the figures before they affect a recommendation.

Translation checks precede the solve. The compiled program is compared with the interview in two ways. The translation kernel of Translation requires its own trusted kernel checks properties that admit exact comparison. A model reviewer separately judges whether the algebra represents the intended meaning. Both checks operate on the program recorded in the ledger. The solve then follows the contract of Untrusted systems propose and trusted kernels decide, with a log distinguishing verified derivation from solver search. The result includes certified marginal values where available, alternative settings and their costs, and validity ranges for the coefficients. Once a plan exists, discrepancies can be classified by their effect. A discrepancy involving a constraint that is active at the recommended plan, or a lever used by that plan, stops the run. A discrepancy involving only unused capacity is reported as a note.

Verifier-guided decision making

VeriPIE enables verifier-guided decision making. A decision compiler turns a natural-language goal into an operational program. It exposes every load-bearing number in a provenance ledger, then sends the program to a solver and trusted-kernel checks. A solver answer is not enough. Model elements must be traceable. The translation must preserve the stated decision. The

proposed setting must be feasible. The kernel must also reproduce the claimed value and optimality proof.

Together, these checks form a feedback loop around the decision compiler. A failed translation check points to a requirement that was lost or invented. An incomplete ledger points to a number with no recorded source. A failed feasibility or optimality check points to a program or solution that cannot be certified. VeriPIE v1.0 implements a bounded version of this loop. After the semantic and exact translation audits, the compiler may make at most two repair attempts. A repair is accepted only if it is a valid program, preserves every represented interview lever and limit, does not increase the number of unpriced limits, and strictly reduces the blocking findings. Accepted and rejected attempts remain in the repair log. A decision owner can then correct or explicitly approve the remaining model elements.

This is inference-time repair and decision review. VeriPIE v1.0 does not update model parameters and does not emit the vector below as a training reward. The current product is therefore neither reinforcement learning with verifiable rewards (RLVR) nor reinforcement learning via symbolic feedback (RLSF). However, its compiler, ledger, translation checker, exact solver and trusted kernel provide signals that a future training system could use.

RLVR and RLSF are related but distinct lines of work that both appeared in 2024. RLSF appeared in May 2024 [17]; the RLVR label followed later that year in an open post-training study.[1] RLVR commonly turns an automatically checkable outcome into a scalar reward. RLSF uses certificates from symbolic tools to provide denser feedback about where a generated formal object fails. Neither direction should be presented as subsuming the other. VeriPIE's proposed feedback vector is closer in spirit to RLSF because it preserves which symbolic obligation failed, while the same checks could also be reduced to an RLVR-style outcome reward.

Continual learning remains future work. A governed deployment could build a versioned training set from accepted and rejected compiler repairs, provenance corrections, failed certificates, approved operational programs, later overrides and observed outcomes. On a fixed review schedule, the organisation could update a private model offline and test it against frozen evaluation cases. Technical and business owners would approve any release. Uncontrolled online self-training would be unsafe because corrections can be wrong, feedback can be manipulated, and policy or data drift can reward the wrong behaviour. The verification kernels must remain independent and separately versioned. Every proposal from an updated model must still pass the current checks. Continual learning may increase the rate of certifiable proposals, but it does not make the proposal model trusted.

A future training system could represent that feedback as a vector rather than one pass or fail value:

$$\mathbf{R} = (R_{\text{parse}}, R_{\text{provenance}}, R_{\text{translation}}, R_{\text{feasibility}}, R_{\text{optimality}}, R_{\text{approval}}) .$$

The components measure six things: whether the program can be reconstructed; whether every influential number has a recorded origin; whether the formal model preserves the stated decision; whether the proposed setting satisfies the constraints; whether the claimed result is exactly optimal; and whether the decision owner approved the model elements on which the result depends. A deployment may combine the components into one RLVR-style reward. Keeping the vector, as in the RLSF direction, prevents different failures from being treated as the same and leaves room for certificate-derived feedback at the program-field or token level.

This direction could support private decision models for high-stakes enterprises. An organisation could start with an open-source language model and adapt it to its own decision vocabulary,

[1] The 2024 source that introduced the RLVR label is available at `https://arxiv.org/abs/2411.15124`.

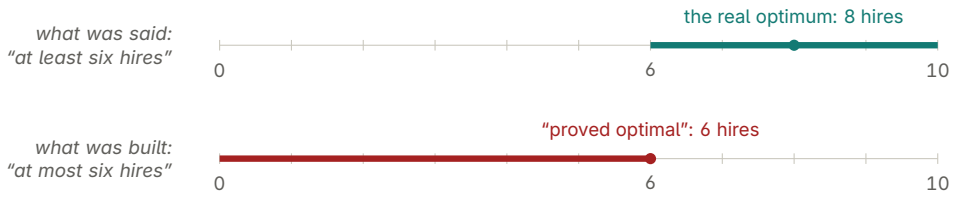


Figure 6: The direction defect. Both programs contain a limit with the number six in it, so every presence check passes; but the floor and the ceiling face opposite ways, and the valid proof is about the wrong feasible set. The gap between the marked points cost two hires and nine hundred thousand dollars.

policies, approved data sources, model corrections and certificate outcomes. The language model would remain an untrusted proposal mechanism. VeriPIE would provide the decision compiler, structured feedback and verification boundary around it. Training data could include rejected translations, corrected ledger entries, failed certificates and approved operational programs. Those records would not need to leave the organisation’s controlled environment.

Fine-tuning on confidential examples would not make the private model trusted. Each important output would still need to pass the same model, provenance, feasibility and optimality checks. This separation matters in finance, healthcare, critical infrastructure, legal operations and other settings where private context matters but fluent text is not proof. Verifier-guided training may help the model propose certifiable decision programs more often. The certificate still determines whether the required checks have closed for a specific decision.

Translation requires its own trusted kernel

As argued in Formal correctness does not establish faithful translation, checking a formal result does not check the natural-language translation that produced it. Earlier versions of the system relied on a model reviewer to compare the interview with the compiled algebra, but its findings did not affect execution. The report could therefore warn that the model might not match the description and still lead with a proved answer.

That asymmetry produced a concrete failure. An interview recorded a floor of at least six hires. The compiled program represented the same lever and value as a ceiling. The resulting plan stopped at six hires, two below the actual optimum for the intended program, with a reported difference of nine hundred thousand dollars. Its optimality proof was valid for the compiled program. The existing checks passed because they confirmed the presence of the lever and the value six but did not compare the direction of the limit. Figure 6 shows the defect.

The translation kernel adds three controls. First, its comparisons are exact: ordinary code compares fields recorded by the interview with fields in the compiled program. Second, its findings affect execution. A violation involving a constraint that is active at the recommended plan, or a lever used by that plan, stops the run. A difference involving only unused capacity is reported as a note. Third, a finding that would stop the run may be overridden only through an explicit recorded decision. The finding remains in the certificate together with the override; it is not converted into a proof. This preserves room for modelling judgement while preventing an unresolved discrepancy from passing silently. Each finding states what the comparison concerns, what the reader provided, and what the program used.

We do not publish the rule inventory, but we describe one class here. A compiler may legitimately rescale a number, such as months into quarters. A cap that changed by a period factor is a scaling, and a cap that changed by anything else is a different number. A rule that is too strict

rejects valid unit conversions. A rule that is too permissive accepts unjustified substitutions. What it may never do is reverse the direction of a constraint: a floor is not a ceiling. One question genuinely cannot be made exact, whether the algebra means what the sentence meant, and that stays with the reviewer, deliberately. The kernel removes from the translation layer the class of error that comparison can settle, so the judgement that remains is smaller, visible, and cannot affect the result without being disclosed.

A fitted formula cannot be verified

Most stages derive results from recorded inputs. Fitting the formula is different because it is an inductive step. A fit cannot be verified. Given eight competitors and their results, the system can compute the affine function that explains them best. No solver can show that the function predicts the ninth case. The system therefore refuses a fit when the result would look authoritative but mean little. This happens when there are too few cases, when a lever never moves, or when levers always move together. In those cases, the data cannot identify or separate the coefficients. The system also reports an exactly determined fit instead of hiding it. If four points force every residual to zero, the perfect fit is an exactly determined interpolation and offers no evidence about a new case.

The system therefore answers a narrower question: how far may a coefficient vary before the recommendation changes?

Proposition 3 (The coefficient validity region is convex)

For a fixed feasible set X and a selected plan $x^\star \in X$, the set of objectives under which $x^\star$ stays optimal,

$$\mathcal{C}(x^\star) \;=\; \big\{\, c \;:\; c^\top (x^\star - x) \;\geq\; 0 \;\text{ for every } x \in X \,\big\},$$

is an intersection of half-spaces, hence convex. Its slice along any coefficient direction, $T_j = \{\, t : c + t\,e_j \in \mathcal{C}(x^\star) \,\}$, is therefore an interval containing 0; certifying two points $t^- \leq 0 \leq t^+$ certifies all of $[\,t^-, t^+\,]$, and an interval reported this way is contained in the true one.

The resulting error is one-sided: the reported range cannot be wider than the true optimality interval. This does not validate the fitted formula. It certifies only a range over which the selected plan remains optimal. Figure 7 illustrates both properties.

The system also checks whether the selected plan remains optimal when several assumptions vary at the same time. Their declared ranges define a rectangular parameter region, and every plan's score is affine in those parameters because each outcome entry is either a literal or a reference to a single parameter.

Proposition 4 (Checking every corner certifies the parameter region)

Let B be the box $[\underline{\theta}_1, \overline{\theta}_1] \times \cdots \times [\underline{\theta}_k, \overline{\theta}_k]$ declared by the assumptions, and suppose every plan's score $s(x, \theta)$ is affine in θ. For each alternative plan x the gap $g_x(\theta) = s(x^\star, \theta) - s(x, \theta)$ is affine, so its minimum over B is attained at a vertex. If $g_x \geq 0$ holds at all 2^k vertices for every alternative x, then $x^\star$ is optimal at every point of B.

The system establishes the proposition's premise by treating each of the 2^k corners as a separate

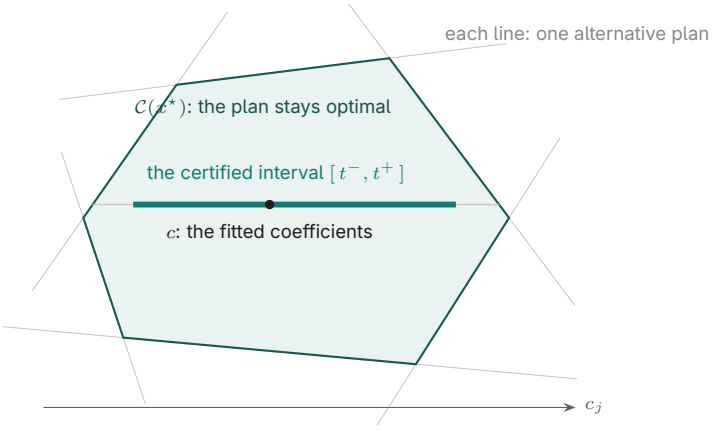


Figure 7: The optimality region for two coefficients. Each alternative plan contributes one straight boundary. The horizontal slice through the fitted point gives the range of one coefficient for which the selected plan remains optimal. If the system certifies only part of the full range, the reported interval is narrower than the true interval, never wider.

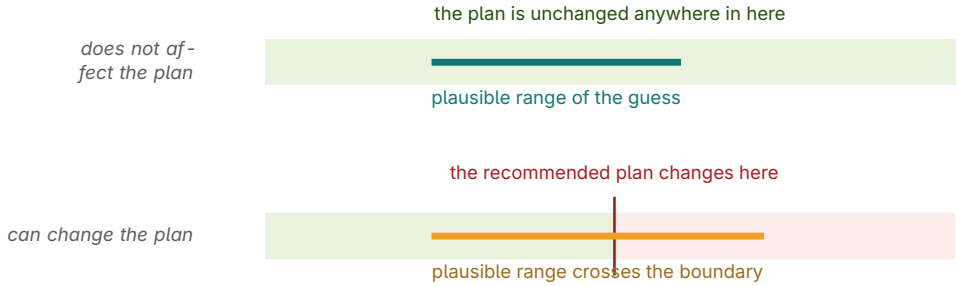


Figure 8: Assumptions are classified by their effect on the recommendation. If the plausible range stays within the certified range, the assumption does not change the selected plan. If it crosses the boundary, the boundary gives the value that should be confirmed.

optimisation instance. At every corner, the selected plan is evaluated in exact arithmetic and must receive the same proved optimality verdict from the arithmetic kernel. A refuted, refused, or solver-trusted corner is insufficient to certify the entire parameter region.

This guarantee requires the feasible set to remain fixed. When a ranged parameter appears in a constraint rather than only in the scores, changing the parameter also changes which plans are feasible. The corner argument no longer applies, so the system reports that it cannot certify robustness for that parameter region.

Together, the plausible range of an assumption and the certified optimality range determine its priority. An assumption is **decision-insensitive** when the same plan remains optimal throughout its plausible range. It is **decision-critical** when the recommendation changes within that range. In the latter case, the boundary identifies a value that should be measured or confirmed before the plan is used (Figure 8). This classification does not improve the estimate itself; it identifies which uncertainty can change the decision.

An unsourced number is not a fact

The system does not treat an unsourced number as a fact. This rule applies to both the observations used to fit coefficients and the figures used to compile limits. An invented case

can change the fitted response. An invented per-unit cost can change which constraint binds. In both places, provenance can change the recommendation rather than only its presentation.

Every quantitative ledger entry has one of five typed origins. It is `stated` when the person supplied it, `sourced` when it came from a retrieved source, fitted when it was measured from recorded cases, `assumed` when the system supplied and labelled it, or missing when the system could not supply a defensible value. Stated, sourced and fitted entries count as facts in the current implementation. An assumption may enter the program, but it does not become a fact. A missing entry has no value and keeps the related obligation unresolved. These origins are stored as typed fields in the program ledger. An earlier reporting branch credited coefficients to the reader even when the system had assumed them, because the attribution existed only as prose. The report and interface are now generated from the recorded origin. A structured field can be tested against the program; an independently written sentence cannot.

When a required aggregate is not published, the system derives it from components instead of asking a model to supply a total. For a business trip to Houston, those components may include airfare, lodging and meals, together with an explicit assumption about the number of nights. Each external component carries a URL or declared range. The model supplies an arithmetic expression over component names, and the kernel evaluates that expression in exact rationals. The generated total therefore remains traceable to the recorded components and assumptions.

The formalisation includes one further check. Models from different families formalise the same problem independently and are compared using a signature that does not depend on their chosen variable names, $\sigma = (\text{class},\ \text{cell},\ v(s^\star) \in \mathbb{Q},\ \text{verdict})$. The first two entries are structural identifiers recorded by the implementation; the remaining entries are the exact objective value and the verification verdict. Formulations that describe the same problem must agree on these entries even if they use different names. Agreement between models from different families is heuristic evidence that the proof concerns the intended problem. Disagreement identifies an ambiguity for the reader to resolve.

A valid proof can certify the wrong program

The following failure analysis describes one reproduced run with its goal and defaults unchanged. It is not a benchmark or an estimate of defect frequency. Its purpose is to show how a valid optimisation result became detached from the user's problem. All four defects arose at one interface: rate questions in the interview had empty defaults, so accepting those defaults recorded no rates.

1. **The report credited the reader with coefficients they never saw:** the provenance sentence above, appended in a branch where the assumptions had just been made. Prose about provenance is not provenance.
2. **A guard switched itself off exactly when it was needed.** The rule catching invented rates was conditioned on the interview recording some rates, and a brief recording none is precisely the run where every coefficient came from somewhere else. A check conditioned on the presence of good input is not a check.
3. **The reviewer was handed contradictory input and blamed for noticing.** It saw an interview with no rates and a formula full of coefficients and reported the obvious, which the system rendered as a mistranslation warning when nothing had been mistranslated. A labelled as-

sumption and an invention must be distinguishable to every consumer downstream, including the ones that only write prose.

4. **A budget was dropped, and this one changed the answer.** A limit with no per-unit cost caps nothing; the rescue that assumes missing costs was gated on every limit being unpriced; the run priced two trivial caps and lost the one limit that made the levers compete. It reported proved optimal, and it was: of a problem with no budget in it.

The four defects belong to one class: the proof was valid for a program that did not preserve the user's input. Solver-side verification cannot detect that class because it receives only the compiled program. The example therefore motivates the translation kernel and shows why the arithmetic kernel is insufficient on its own. These defects were found in August 2026 by reproducing a run after the system had reported a proved answer. The example does not establish that the defect class is complete.

The benchmark shows process gains, not better decisions

The formal results in this paper are conditional soundness claims. When the arithmetic kernel reports *proved*, the stated certificate conditions hold for the serialised program. The sensitivity results also establish properties of that program under their stated conditions. Neither result shows how often the natural-language translation is correct or how often a user receives a useful plan.

KNOWIDEA evaluated VeriPIE on a private consulting case-study benchmark with 20 operational questions and 40 scored runs. This section is the public benchmark summary. The questions were drawn from real use cases in KNOWIDEA client work across oil and gas, finance, manufacturing, and other industries. They are represented as de-identified synthetic scenarios so the evaluation does not publish client records or reproduce any one engagement in full. Client names, organisations, dates, raw records and identifying quantities are not included, and no client endorsement is implied.

The VeriPIE arm used a state-of-the-art language model inside the full product pipeline. The SOTA-LLM baseline used a direct conversational setup. Both arms received the same opening line, could question the same instrumented simulated user, and ended with the same request for a final recommendation. A separate state-of-the-art language model scored both outputs. Each case was run once, and the retained log bundle reports no benchmark errors. Full prompts, transcripts, client-derived case materials and implementation source are proprietary and are not part of the public release.

The final row distinguishes a machine-checkable certificate from a written recommendation. VeriPIE records the declared program and independently checks feasibility and selected optimality claims before issuing the certificate. The direct SOTA-LLM baseline does not produce that certificate artifact. This distinction does not show that the declared program is the right business model; the limits in Verification does not make the model correct still apply.

The elicitation score is the share of predefined load-bearing facts that an arm asked for or received. Assumption disclosure counts unsupplied figures that the final answer explicitly labels as assumptions. Stated origin counts figures assigned a source category in the scored output. Constraint handling checks whether extracted limits are respected. Sensitivity counts

Measure	VeriPIE	SOTA-LLM
Mean load-bearing fact elicitation	69.0%	49.3%
Unsupplied figures disclosed as assumptions	53.1%	9.9%
Figures with a stated origin	76.7%	44.2%
Stated constraints respected	73.7%	74.3%
Plans with sensitivity thresholds	90.9%	75.0%
Plans naming unknowns	100%	95%
Machine-checkable verification certificate produced	✓	×

Table 3: VeriPIE and SOTA-LLM on KNOWIDEA's 20-case operational benchmark. Figure, provenance, constraint and sensitivity rates use completed plans only. Elicitation uses all completed runs. The final row is a binary system-capability distinction, not a benchmark percentage.

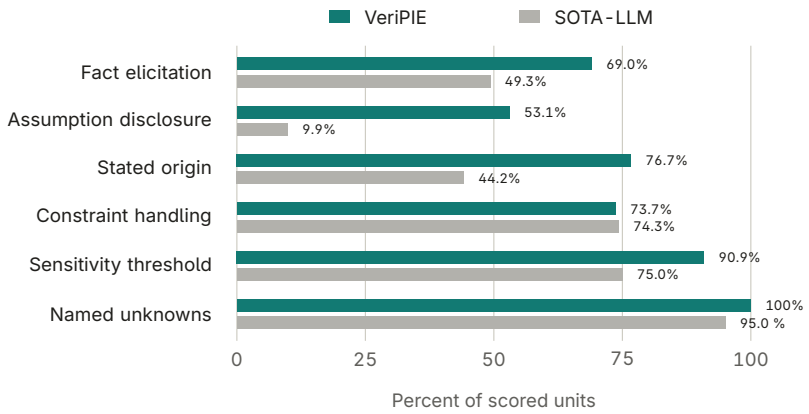


Figure 9: Process measures reported in the 20-case benchmark. The bars compare different denominator units and are descriptive, not estimates of business decision quality.

completed plans that state a threshold at which the recommendation changes. These definitions are published so the aggregate comparison can be interpreted even though the underlying product and case records remain closed.

Because every case was run once, the percentages are descriptive results for this benchmark, not population estimates. Figure-level events within the same plan are correlated, so treating them as independent trials would overstate certainty. The paper therefore does not report confidence intervals for this first run.

The comparison supports only a narrow claim. VeriPIE elicited more load-bearing facts, labelled more unsupplied figures, recorded more origins, and reported sensitivity thresholds more often. Constraint handling was effectively tied. The benchmark therefore measures process discipline and inspectability, not a general claim that VeriPIE is better than state-of-the-art language models.

The benchmark does not score whether a recommendation is a good business decision. The retained report names two counterexamples. One marketing model passed the current gates while making *do nothing* win because the objective represented activity as a cost without its revenue tradeoff. One collections plan violated an extracted constraint. The comparison is also not model-matched, uses one repeat, relies on a model extractor for figure-level attributes, and compares against a direct conversational baseline rather than a tool-using product.

A separate solver-focused suite retained by KNOWIDEA rules out another tempting claim. A

Metric	Denominator	Example of a positive score
Fact elicitation	Predefined load-bearing facts across all 20 runs	The arm asks for a budget cap that the case defines as load-bearing.
Assumption disclosure	Unsupplied quantitative figures in completed plans	A forecast added by the arm is explicitly labelled as an assumption.
Stated origin	Quantitative figures in completed plans	A hiring figure is linked to supplied data, a fitted model or a stated assumption.
Constraint handling	Extracted stated constraints in completed plans	The recommended allocation remains within the stated budget cap.
Sensitivity	Completed operational plans in each arm	The answer states a coefficient threshold at which the plan changes.
Named unknowns	Completed operational plans in each arm	The answer names a missing estimate that could change the decision.

Table 4: Denominator units and scoring examples for the reported process measures. Figure-level measures contain different numbers of observations in different cases.

warned frontier model with a sandbox matched VeriPIE on the tested optimisation traps. On hard floating-point MILP near-ties, VeriPIE could also return a wrong candidate, but the exact kernel did not certify it. Across the reported solver families, the kernel-verified error count remained zero. The defensible advantage is therefore a machine-checkable guarantee when the kernel closes, not a claim that VeriPIE always finds a better optimum.

Future evaluation should add a model-matched baseline, at least three repeats, an objective-contract audit, dimensional checks, and independently reviewed reference programs. Independent technical review should also audit the certificate checker and trusted computing base against the claims in this paper. Evaluation should use multiple reviewers and report their agreement. It should also measure override frequency, latency, cost, and how often the translation kernel prevents a sound certificate from being attached to the wrong operational question.

VeriPIE extends trusted-kernel design to model translation

The trust architecture draws on established work. An untrusted search procedure emitting a certificate that a small trusted program checks is the organising principle of LCF-style proof assistants [14] and proof-carrying code [20], and standard practice in modern solving, where SAT solvers emit proofs a separate checker validates [25] and exact rational certificates exist for linear and integer programming [2, 9, 11]. Those certificates are stronger than ours; VIPR validates a full branch-and-bound derivation where we ask only for a bound any dual vector can witness, generality traded for a checker small enough to read in an afternoon. Autoformalization and symbolic regression are treated in Formal correctness does not establish faithful translation as conceptual analogies. We do not claim that either literature directly addresses operational model translation.

The work on language models for optimisation modelling [1, 21] is largely about getting a correct model from a description, evaluated against a reference. We take the complementary position: the model will sometimes be wrong regardless, and the useful engineering makes the wrongness detectable and consequential at run time, on problems that have no reference. Frameworks placing a language model inside a loop of external critics [18] are the closest framing. Our specific contribution is an exact, rather than model-based, translation critic whose findings bind the result instead of merely annotating it. The validity ranges and the robustness

box are classical operations research, sensitivity analysis and robust optimisation [7, 8]. VeriPIE uses them as the designated response to an unverifiable induction step.

Verification does not make the model correct

The fitted formula cannot be verified from the observations used to construct it. This paper certifies sensitivity ranges around that fit; it does not prove that the fit predicts new cases. The kernel also cannot decide whether the compiled algebra captures the full meaning of the original description. A model reviewer retains that responsibility, although the exact checks reduce the set of discrepancies left to judgement.

Robustness is refused whenever a ranged parameter changes the feasible set. This restriction applies to a meaningful class of operational problems. The allocation may also be proved while its implementation schedule remains a planning judgement. For example, the program may support a conclusion of eight hires without establishing that three should be hired in the current quarter. The report labels this boundary so the authority of the certificate does not extend to an uncertified schedule.

Every optimal verdict is conditional on a declared weight vector. The system therefore also reports a weaker, weight-free Pareto claim: whether another feasible plan is at least as good on every axis and better on one. Finally, the four defects described above were found in August 2026. They should not be read as a complete inventory of translation failures.

The benchmark adds a practical limit. An internally traceable objective can still encode the wrong decision, as the marketing counterexample shows. KNOWIDEA's internal MILP stress tests also show that a floating-point search may miss the true optimum. In that case, the safe claim is that the kernel refused to certify the result, not that the system found the right plan.

Enterprise deployment requires controls beyond the certificate

A decision certificate does not replace enterprise security, governance or operating controls. Before deployment, KNOWIDEA and the customer must define data residency, retention and deletion, model-provider access, identity and access controls, audit logging, reviewer and override roles, escalation, source-system integrations, monitoring and incident response. These controls determine whether the system is suitable for a particular organisation even when the formal checks work as described.

A practical deployment should pass four separate gates. The table describes the evidence that an enterprise should require; it does not claim that a certificate satisfies these obligations by itself.

The human approval boundary must also be explicit. A *verified decision* should not be released until an accountable owner confirms that the objective, levers, limits and fitted effects represent the decision being made. Refusals, manual overrides and later corrections should remain in the audit record so an organisation can review how decisions changed over time.

Hosting arrangements, service levels, integrations, implementation schedules and pricing are commercial terms. They are not established by this whitepaper and must be defined for each deployment. The certificate addresses the stated decision program; enterprise readiness

Gate	Main question	Evidence required
Qualify	Is the decision measurable and bounded?	Written decision scope, named owner, intended use and explicit exclusions.
Configure	Can data and model access meet policy?	Approved data flow, retention rules, access controls, provider terms and integration design.
Validate	Does the configured system behave acceptably?	Representative test cases, acceptance criteria, model review, security review and recorded limits.
Operate	Can the organisation govern change and failure?	Audit trail, override and escalation rules, monitoring, incident response and review cadence.

Table 5: A customer adoption path outside the mathematical certificate.

depends on both that certificate and the surrounding controls.

Verification must cover the model before the solve

A solver establishes properties of the model it receives. VeriPIE focuses on the engineering that decides which model reaches the solver. This includes the interview that records levers and limits, the provenance system that classifies every quantitative input, the fitting stage that refuses unidentifiable coefficients, and the translation kernel that compares the compiled program with the recorded description.

The current system can certify arithmetic, feasibility, selected optimality claims, coefficient ranges, and exact translation properties. It cannot prove that an induced formula predicts new observations or that every modelling choice captures the user's intent. Its contribution is to make that boundary operational. Checkable claims may be reported as proved; assumptions and judgements remain visible; and a failed obligation can stop the run before a valid certificate lends authority to the wrong program. The benchmark suggests that this discipline improves elicitation, disclosure, provenance and sensitivity reporting, while also exposing unresolved model faithfulness.

References

[1] Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Optimization modeling using MIP solvers and large language models, 2023.

[2] David L. Applegate, William Cook, Sanjeeb Dash, and Daniel G. Espinoza. Exact solutions to linear programming problems. *Operations Research Letters*, 35(6):693–699, November 2007. ISSN 0167-6377. doi: 10.1016/j.orl.2006.12.010.

[3] Associated Press. Deloitte to partially refund australian government for report with apparent ai-generated errors. Associated Press, 2025. URL https://apnews.com/article/ab54858680ffc4ae6555b31c8fb987f3. Accessed 16 August 2026.

[4] Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, et al. ProofNet: Autoformalizing and formally proving undergraduate-level mathematics, 2023.

[5] Clark Barrett and Cesare Tinelli. *Satisfiability Modulo Theories*, pages 305–343. Springer International Publishing, 2018. ISBN 9783319105758. doi: 10.1007/978-3-319-10575-8_11.

[6] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954. ISSN 0273-0979. doi: 10.1090/s0002-9904-1954-09848-8.

[7] A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Mathematics of Operations Research*, 23(4):769–805, November 1998. ISSN 1526-5471. doi: 10.1287/moor.23.4.769.

[8] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations Research*, 52(1):35–53, February 2004. ISSN 1526-5463. doi: 10.1287/opre.1030.0065.

[9] Kevin K. H. Cheung, Ambros Gleixner, and Daniel E. Steffy. *Verifying Integer Programming Results*, pages 148–160. Springer International Publishing, 2017. ISBN 9783319592503. doi: 10.1007/978-3-319-59250-3_13.

[10] Leonardo de Moura and Nikolaj Bjørner. *Z3: An Efficient SMT Solver*, pages 337–340. Springer Berlin Heidelberg, 2008. ISBN 9783540788003. doi: 10.1007/978-3-540-78800-3_24.

[11] Leon Eifler and Ambros Gleixner. A computational status update for exact rational mixed integer programming. *Mathematical Programming*, 197(2):793–812, January 2022. ISSN 1436-4646. doi: 10.1007/s10107-021-01749-5.

[12] Financial Times. EY retracts study after researchers discover AI hallucinations. Financial Times, 2026. URL https://www.ft.com/content/a61cbcae-95e4-4449-86e1-ef40fb306f4e. Accessed 16 August 2026.

[13] Financial Times. KPMG report contained AI hallucinations on benefits of AI. Financial Times, 2026. URL https://www.ft.com/content/b3828e92-4961-4b39-84f0-c42f33be3c3f. Accessed 16 August 2026.

[14] Michael J. Gordon, Arthur J. Milner, and Christopher P. Wadsworth. *Edinburgh LCF*. Springer Berlin Heidelberg, 1979. ISBN 9783540385264. doi: 10.1007/3-540-09724-4.

[15] Kurt Gödel. An example of a new type of cosmological solutions of Einstein's field equations of gravitation. *Reviews of Modern Physics*, 21(3):447–450, July 1949. ISSN 0034-6861. doi: 10.1103/revmodphys.21.447.

[16] Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, December 2017. ISSN 1867-2957. doi: 10.1007/s12532-017-0130-5.

[17] Piyush Jha, Prithwish Jana, Pranavkrishna Suresh, Arnav Arora, and Vijay Ganesh. RLSF: Reinforcement learning via symbolic feedback, 2024.

[18] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, et al. LLMs can't plan, but can help planning in LLM-Modulo frameworks, 2024.

[19] Leonardo de Moura and Sebastian Ullrich. *The Lean 4 Theorem Prover and Programming Language*, pages 625–635. Springer International Publishing, 2021. ISBN 9783030798765. doi: 10.1007/978-3-030-79876-5_37.

[20] George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '97*, POPL '97, pages 106–119. ACM Press, 1997. doi: 10.1145/263699.263712.

[21] Rindranirina Ramamonjison, Timothy T. Yu, Raymond Li, Haley Li, et al. NL4Opt competition: Formulating optimization problems based on their natural language descriptions, 2023.

[22] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, April 2009. ISSN 1095-9203. doi: 10.1126/science.1165893.

[23] Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16), April 2020. ISSN 2375-2548. doi: 10.1126/sciadv.aay2631.

[24] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, et al. Self-consistency improves chain of thought reasoning in language models, 2022.

[25] Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. *DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs*, pages 422–429. Springer International Publishing, 2014. ISBN 9783319092843. doi: 10.1007/978-3-319-09284-3_31.

[26] Norbert Wiener. Some moral and technical consequences of automation: As machines learn they may develop unforeseen strategies at rates that baffle their programmers. *Science*, 131(3410):1355–1358, May 1960. ISSN 1095-9203. doi: 10.1126/science.131.3410.1355.

[27] Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In *Advances in Neural Information Processing Systems 35*, NeurIPS 2022, pages 32353–32368. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2022. doi: 10.52202/068431-2344.

[28] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. MiniF2F: A cross-system benchmark for formal Olympiad-level mathematics, 2021.