ORSMT: UNLEASHING THE POTENTIAL OF SATISFIABILITY MODULO THEORIES FOR SEMANTIC VERIFICATION AND REPAIR IN LLM-BASED OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models increasingly automate optimization modeling from natural language problems, producing written mathematics, solver code and a returned report. Verifying one optimization problem across these artifacts requires semantic comparison: unequal models can attain the same objective, equivalent models can use different auxiliary variables, and mutually consistent artifacts can omit the same source requirement. ORSMT provides a typed semantic interface and exact SMT relation queries. It aligns observable decisions, projects auxiliary variables, proves or refutes feasible region and objective relations, localizes refutations to implicated rows or domains and links source texts when available. It then reconstructs one artifact from another, and each adopted candidate passes the required relation checks. Across 3,034 problems and six generator sizes, ORSMT improves the model–code match rate in all 72 benchmark and generator combinations before selection. Protected selection raises the aggregate rate from 43.2% to 88.2% and the share that are both correct and matched from 25.2% to 40.2%. On the 1,905 refuted outputs, certified voting under the shared fallback makes 50.9% of outputs correct and matched, compared with 44.8% for ReLoop and 49.4% for Opt-Verifier. A separate one-round source revision followed by exact reconstruction restores a model–code match for 84.4% of the refuted pairs. Code and frozen core artifacts are available at `https://anonymous.4open.science/r/ORSMT-7083/`.

1 INTRODUCTION

Optimization models support logistics, production planning, and energy scheduling by optimizing decisions under operational constraints. Decision quality depends on how natural-language requirements are translated into mathematical formulations, executable solver models, and reported decisions. Recent large language model (LLM) systems generate these artifacts together, while evaluation commonly focuses on solver execution and the final objective value. Establishing **semantic agreement across these artifacts** therefore motivates verification and repair at the level of the optimization problem itself.

Classical criteria such as feasibility and answer correctness evaluate the returned decision without assessing the agreement across the underlying artifacts (Ramamonjison et al., 2022; AhmadiTeshnizi et al., 2024b; Huang et al., 2025a; Jiang et al., 2025; Yang et al., 2025). Different feasible regions can attain the same optimum, so a correct objective value does not identify the implemented problem. Conversely, equivalent models can differ in auxiliary variables, redundant constraints, and syntax. Agreement between written mathematics and executable code can also miss a source requirement from the natural language omitted by both. Output-based evaluation cannot distinguish these failures or establish whether a proposed repair restores the required relations.

Recent methods broaden feedback beyond the final value through behavioral tests, structural comparisons, model-based judgments, and counterexample searches (Zhang et al., 2026; Li & Hai, 2026; Zhai et al., 2025; Wang et al., 2025; Lian et al., 2026; Liu et al., 2026). Although these methods provide useful repair signals, passing behavioral tests or receiving a favorable model-based judgment does not establish semantic agreement between the math formulation and its code implementation. A program can produce the expected outputs on tested cases while admitting decisions excluded by the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

formulation, whereas structural comparisons can reject equivalent models that use different auxiliary constructions. Verification must therefore examine whether the different artifacts admit the same feasible decisions and preserve their objective preferences. It should also localize discrepancies to guide repair and verify that the repaired artifacts satisfy the required relations before adoption.

This paper proposes **ORSMT**, a satisfiability modulo theories (SMT)-based verification and repair framework for source requirements, written mathematics, instantiated solver models, and returned reports. ORSMT translates the written model and captures the solver model from code in a common exact representation, aligning observable decisions and projecting auxiliary variables. Within the affine rational core, SMT queries check admitted source requirements, feasible regions, objective relations, and reports for a fixed instance. A satisfiable violation query returns a localized counterexample; an unsatisfiable query certifies the corresponding relation. The evidence supports direct reconstruction between mathematics and code or guides revision, and every adopted candidate passes the required checks again. This keeps diagnosis, repair, and acceptance as distinct stages of one formally checked process. Our contributions are summarized as follows:

(i) We introduce ORSMT, a semantic verification framework for source requirements, mathematical formulations, solver codes, and returned reports. Its common representation supports exact SMT checks within the admitted affine rational core, providing certificates or localized counterexamples for decided relations.

(ii) We develop a verification and repair loop that combines bidirectional reconstruction with counterexample-guided revision. Each adopted candidate passes the required relation checks, connecting diagnosis, repair, and acceptance through the same verifier.

(iii) We evaluate ORSMT on 3,034 problems from 12 benchmarks and 6 generator sizes. Model–code match rates improve in all 72 benchmark–generator combinations before selection. Under protected selection, the aggregate match rate rises from 43.2% to 88.2%, and the share of outputs with both a correct answer and a model–code match rises from 25.2% to 40.2%. ORSMT further improves correct-and-matched outputs when integrated with existing OptiMUS-0.3 and OptiMind pipelines, supporting its use as a modular verification and repair layer.

2 Related Work

Autoformulation and evaluation. NL4Opt (Ramamonjison et al., 2022) studies formulation. OptiMUS and OptiMUS-0.3 build modular formulation and execution pipelines (AhmadiTeshnizi et al., 2024b;a), while OptiMind adds expert-guided training and inference (Zhang et al., 2025). ORLM, LLMOPT and ORThought generate and revise mathematical models, code and solver outputs (Huang et al., 2025a; Jiang et al., 2025; Yang et al., 2026). MAMO, OptiBench and OPT Engine broaden evaluation tasks (Huang et al., 2025b; Yang et al., 2025; Chen et al., 2026). Feasibility and reference objectives assess final answers, but do not establish that the written model, instantiated code and report describe the same optimization problem.

Verification and repair. ReLoop tests program behavior under parameter changes (Lian et al., 2026). Opt-Verifier judges structure and solutions (Liu et al., 2026), and SAC Opt uses semantic anchors for revision (Zhang et al., 2026). EquivaMap and ORGEval compare variable mappings or graph structure (Zhai et al., 2025; Wang et al., 2025). Falsification uses solver-based testing under controlled instance transformations, whereas FLARE verifies MILP reformulations using theorem proving (Li & Hai, 2026; Robbins et al., 2026). ORSMT repairs instance-specific mathematical, executable and report artifacts. Therefore, repair comparisons use ReLoop and Opt-Verifier.

Formal semantic verification. Denotational tests evaluate program behavior beyond textual similarity (Zhong et al., 2020). Z3 and cvc5 decide arithmetic formulas (de Moura & Bjørner, 2008; Barbosa et al., 2022), while neural symbolic methods translate proposed statements into formal queries (Jha et al., 2023; Pan et al., 2023; Ye et al., 2023; Olausson et al., 2023). ORSMT verifies source requirements, feasible regions, objective preferences and reports in this way. Its evidence identifies missing, additional and redundant restrictions and supports repair from either usable model.

3 PROBLEM SETTING

An autoformulation pipeline maps a source statement s and instance data to written mathematics, a solver program $\widehat{P}$, and a report $R = (\hat{x}, \hat{z}, \text{status})$, where $\hat{x}$ and $\hat{z}$ represent the returned solution and its objective value. The written mathematics is lowered to the mathematical model A. The instantiated solver model is represented by $B = \mathcal{H}(\widehat{P})$. A source adapter supplies formal requirements $C = \mathcal{F}_{\text{src}}(s)$, each linked to its supporting source text and variable roles, when that adapter admits the source. Verification checks the available relations among C, A, B, and R.

Define a model as $A = (X_A, U_A, \Phi_A, f_A, \delta_A)$, where X_A contains observable decisions, U_A contains auxiliary variables, Φ_A contains domains and constraints, f_A is the objective, and $\delta_A \in \{\min, \max\}$. The executable model $B = (X_B, U_B, \Phi_B, f_B, \delta_B)$ is defined in the same way. A fixed correspondence aligns the observable decisions of the two models. Their comparison projects out the auxiliary variables. The projected feasible region and objective graph are

$$F_A^X = \{x : \exists u_A \ \Phi_A(x, u_A)\}, \qquad G_A^X = \{(x, z) : \exists u_A \ \Phi_A(x, u_A) \wedge z = f_A(x, u_A)\}. \quad (1)$$

When G_A^X is single-valued, $v_A(x)$ denotes the unique objective value associated with $x \in F_A^X$.The relation $x \succ_A y$ denotes a strict improvement according to δ_A, and $x \succeq_A y$ also permits equal values. Therefore we define semantic model agreement as

$$A \equiv_{\text{sem}} B \quad \iff \quad (F_A^X, \succ_A) = (F_B^X, \succ_B). \quad (2)$$

This relation preserves feasible decisions and their preference order. Absolute objective values are checked separately. Source satisfaction requires a feasible model that entails every applicable requirement in a nonempty admitted C. Untranslated source units retain an undecided source verdict rather than a vacuous certificate. The report R passes when its returned decision is feasible, correctly valued and optimal. Instance parameters are substituted before comparison, and partial reports use existential completion over unreported coordinates.

The task is to return matched artifacts and evidence for each required relation. A mathematical model can supply a missing implementation, a captured solver model can supply missing mathematics, and a source counterexample can expose an omission shared by both. Every decided relation is certified or accompanied by a concrete refuting assignment.

4 ORSMT: VERIFICATION, REPAIR AND MATCHING

ORSMT first translates the mathematical model A and the executable model B into a common exact representation, then uses SMT to check their relations, and finally applies the same checks to every repaired candidate before adoption.

Figure 1 separates two models translation, SMT checking and the two repair routes. The verifier supports affine models with rational coefficients over real, integer, and Boolean variables. Appendix B.4 gives the complete verification and repair procedure.

4.1 MODEL REPRESENTATION AND BIDIRECTIONAL CONSTRUCTION

Mathematical translation. The translator $\mathcal{L}$ lowers admitted mathematical formulation into the model representation A defined in Section 3. Structured declarations, solver objects, LP/MPS files provide variables, contraints and objectives, while mathematical prose is transcribed together with links to the source text from which each element is derived. Finite decimal coefficients are represented as exact rationals. The resulting representation explicitly records variable bounds and domains, contraints, and contraint rows. Appendix A.2 presents the detailed input formats.

Executable model and correspondence. ORSMT aligns the observable decisions X_A and X_B of A and B through a one-to-one correspondence based on normalized scalar or index-tuple identifiers, variable roles, and types. Auxiliary variables U_A and U_B are not matched directly: they are projected out when comparing the model objects in Eq. (1). A contradiction, type mismatch, or unmatched observable decision prevents a syntactic correspondence from being accepted and leaves the corresponding relation UNDECIDED. This separation allows semantically equivalent models to use different auxiliary constructions or redundant constraints without requiring a one-to-one match of all solver variables.

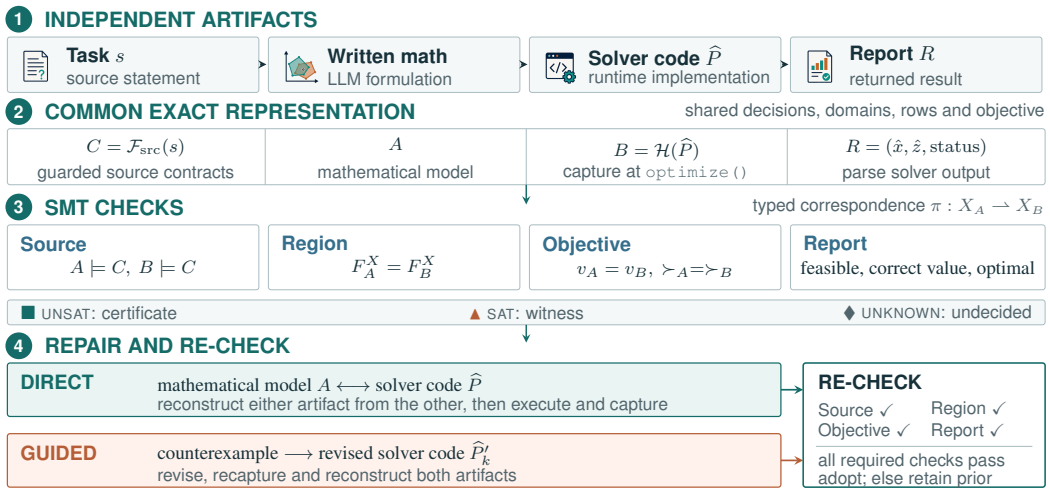


Figure 1: ORSMT translates source requirements, written mathematics, solver code and reports into a common exact representation. SMT checks either certify their relations or return a counterexample for direct reconstruction or revision. After repair, every relaton is checked again before adoption.

Target and normalization. ORSMT supports reconstruction in both directions. When the mathematical model A is available, ORSMT uses it as the primary construction target; otherwise, it uses the captured executable model $B = \mathcal{H}(\widehat{P})$ as the target for reconstructing missing mathematical artifacts. Before construction, ORSMT applies a normalization operator ($\mathcal{N}$) that preserves the projected feasible region and objective graph relevant to the observable decisions. The normalization may make implicit domains explicit and derive redundant constraints or auxiliary definitions needed for compilation, while preserving the semantic model objects used for verification. The resulting normalized representation is then used as the construction target.

Mathematics to program. For the mathematical target $T = A$, ORSMT computes

$$\widetilde{A} = T, \qquad \widetilde{P} = \mathrm{Compile}(\mathcal{N}(T)), \qquad \widetilde{B} = \mathcal{H}(\widetilde{P}), \qquad \widetilde{R} = \mathrm{Solve}(\widetilde{P}). \tag{3}$$

The generated solver code declares the normalized model and reports its decision, objective value, and status. ORSMT then captures the instantiated executable model ($\widetilde{B}$) from the generated program and verifies its semantic agreement with ($\widetilde{A}$), together with the correctness of the returned report ($\widetilde{R}$). Consequently, this direction can be used either to synthesize missing solver code from a mathematical model or to repair existing code that fails to match the available formulation.

Program to mathematics. When the executable model is available but the mathematical formulation is missing or incomplete, ORSMT sets (T=B). It exports the domains, bounds, constraints, objective, and optimization direction of B into a mathematical representation ($\widetilde{A}$), and uses this representation as the construction target. The resulting artifacts are compiled, executed, and captured as in Eq. (3), after which the same verification queries are applied to certify their correspondence. Thus, either representation can serve as the construction source, while both directions return to the same semantic verification procedure.

Proposition 1: Artifact Construction. If the normalization operator $\mathcal{N}$ preserves the projected feasible region and objective graph of its input, and that $\mathcal{H}$ faithfully captures the model compiled from the normalized representation, Eq. (3) constructs artifacts whose projected model semantics agree with the construction target, up to the fixed correspondence of observable decisions. Appendix B gives the full proof and the guarantees for constraint reduction.

4.2 Verification Queries and Diagnosis

Feasible regions. ORSMT encodes domains and rows as exact arithmetic formulas and computes the two directional violation queries

$$Q_{A \setminus B} = \mathrm{SAT}(F_A^X \wedge \neg F_B^X), \qquad Q_{B \setminus A} = \mathrm{SAT}(F_B^X \wedge \neg F_A^X). \tag{4}$$

A witness in $A \setminus B$ identifies a mathematical decision excluded by the implementation. A witness in $B \setminus A$ identifies an implemented decision excluded by the mathematics. Unsatisfiability in both directions establishes equality of the projected feasible regions which includes the case where both regions are empty. Therefore, model feasibility is checked separately before certifying semantic agreement.

Verdict and diagnosis. Each verification query returns either a certificate or a counterexample. A satisfiable violation query refutes the corresponding relation and returns a concrete witness, whereas an unsatisfiable query certifies that the relation holds. After substituting instance parameters, ORSMT extracts the implicated constraints, domains, and linked source texts from the witness and, when applicable, the unsatisfiable core. Objective diagnoses additionally record the evaluated preference relation, while source diagnoses identify the violated requirement and its applicable scope. The resulting diagnostic payload is used for deterministic reconstruction or counterexample-guided language-model revision.

Objective relations. For decisions that are feasible in both models, ORSMT verifies whether the two objectives induce the same preference ordering. It decides the directional violation query:

$$Q^{\text{pref}} = \text{SAT}\big(x, y \in F_A^X \cap F_B^X \wedge \\ [(x \succ_A y \wedge y \succeq_B x) \vee (x \succ_B y \wedge y \succeq_A x)]\big). \tag{5}$$

A satisfying pair exposes opposite ordering or a strict improvement that becomes a tie. Feasible models and unsatisfiable relation queries certify Eq. (2). Absolute values use

$$Q^{\text{value}} = \text{SAT}\big(x \in F_A^X \cap F_B^X \wedge v_A(x) \neq v_B(x)\big). \tag{6}$$

Unsatisfiability certifies equal values at common decisions, separately from preference agreement.

Source requirements. Each source requirement in C is represented as a guarded contract (g_k, ψ_k, S_k), consisting of an applicability condition, a logical obligation, and its supporting source text. Variable identifiers are resolved through the typed source schema before a contract is admitted. Unresolved identifiers or untranslated source units remain undecided, and a source certificate is issued only when the admitted contract set is nonempty. For a feasible candidate, ORSMT decides

$$Q_{A,k}^{\text{source}} = \text{SAT}(\Phi_A \wedge g_k \wedge \neg\psi_k), \qquad Q_{B,k}^{\text{source}} = \text{SAT}(\Phi_B \wedge g_k \wedge \neg\psi_k). \tag{7}$$

Unsatisfiability certifies that the guarded requirement is enforced whenever its applicability condition holds, whereas a satisfying assignment returns a violating decision. A shared witness on both artifacts identifies a common omission and links it to S_k. This source relation is independent of pairwise model equivalence: two artifacts may agree semantically while violating the same source requirement.

Missing, additional and redundant constraints. Row entailment applies the same violation test to individual mathematical and executable restrictions. A written row violated by an implemented decision is missing from the implementation. An executable row violated by a written decision adds a restriction. To test redundancy of a row c, let D_A collect domains and K_A collect constraint rows. ORSMT asks

$$Q_{A,c}^{\text{redundant}} = \text{SAT}(D_A \wedge \bigwedge_{d \in K_A \setminus \{c\}} d \wedge \neg c). \tag{8}$$

Unsatisfiability permits deletion of c. Subsequent tests use the reduced model. A refuting assignment is completed against the entire candidate before revision, and infeasible models receive conflict diagnoses. Appendix C details the reduction and scalability.

Reported result. Let $O \subseteq X_B$ be the reported coordinates and $\tau \geq 0$ the declared reporting tolerance. Let $B_\tau(\hat{x}_O)$ as their tolerance box denote the corresponding tolerance box, with discrete coordinates fixed to admissible integers; coordinates outside O are existentially completed. For minimization, ORSMT computes

$$Q_\tau^{\text{extend}} = \text{SAT}(\Phi_B \wedge B_\tau(\hat{x}_O)),$$
$$Q_\tau^{\text{bind}} = \text{SAT}(\Phi_B \wedge B_\tau(\hat{x}_O) \wedge |f_B - \hat{z}| \leq \tau), \tag{9}$$
$$Q_\tau^{\text{better}} = \text{SAT}(\Phi_B \wedge f_B < \hat{z} - \tau). \tag{10}$$

Maximization reverses the last inequality. Unsatisfiability of the extension and binding queries certifies, respectively, that no feasible completion matches the reported decision and value, while satisfiability provides a completion witness. An unsatisfiable better-decision query certifies that no feasible decision improves on $\hat{z}$ beyond the declared tolerance.

4.3 CERTIFICATES AND SCALABLE EXACTNESS

Relation certificate. With fixed observable correspondence, feasible models, and single-valued objective graphs, unsatisfiability of both region-difference queries in Eq. (4) and the preference query in Eq. (5) certifies $A \equiv_{\text{sem}} B$. A satisfying assignment to any query refutes the corresponding relation. The certificate covers all decisions admitted by the encoded models, while a refutation provides the separating decision or decision pair used for diagnosis. Appendix B proves this claim and the corresponding report guarantees.

Sound reduction. Using the domain formula D_A and active row set K_A defined above, consider a compiled obligation ψ and any $L \subseteq K_A$:

$$\text{UNSAT}(D_A \wedge \bigwedge_{c \in L} c \wedge \neg\psi) \implies \Phi_A \models \psi. \tag{11}$$

If the reduced query is unsatisfiable, the full violation query is also unsatisfiable, certifying the obligation. If it is satisfiable, we test whether its assignment can be extended to satisfy the omitted rows. Only an extension that satisfies the full model and violates ψ is reported as a counterexample; otherwise, the reduced query alone does not refute the obligation.

Exact component slicing. The bipartite row–variable incidence graph retains every connected component containing variables of ψ. On a feasible base model, this slice is exact because disconnected components admit independent satisfying assignments. Smaller neighborhood slices retain the one-way guarantee in Eq. (11). Appendix B gives both proofs.

4.4 REVISION AND SELECTION

Revision. Let $Y_0 = (A, \widehat{P}, R)$ be the current artifacts. Verification evidence e records the failed relation, a counterexample when available, implicated rows or domains, and linked source text. Given s and e, the reviser proposes a program $\widehat{P}'_k$. ORSMT captures its executable model, reconstructs the artifacts through Eq. (3), and rechecks the required source, region, objective, and report relations.

The evidence guides revision: a decision in the mathematical region but not the executable region must be admitted by the implementation, while a decision only in the executable region must be excluded. Objective and source revisions address the violated preference and linked requirement, respectively. For source revision, plan review checks the model against the source through the computed plan; row review checks individual requirements to localize the mismatch.

Selection. Only candidates passing every check required by the active policy are eligible. Direct selection takes the first eligible candidate in a fixed order. Certified voting groups eligible candidates by verified reported value and selects from the largest group if at least two agree. Protected selection also preserves the original finite result when available. If no candidate qualifies, ORSMT retains Y_0; otherwise, it adopts the selected model, program, report, and verification evidence together. Reference objectives and statuses are used only for evaluation. Appendix B.4 gives the complete procedure.

Proposition 2: Certified Adoption. Every adopted candidate satisfies the checks required by its selection policy; an undecided required check prevents adoption. Otherwise, the original artifacts remain unchanged. Voting additionally certifies agreement of at least two eligible candidates on the reported value, and protected selection preserves an available original finite result.

Checks follow their prerequisites: admission and feasibility precede semantic comparison, region agreement precedes objective preference, and report binding precedes optimality. An undecided prerequisite leaves its dependent check undecided.

Certificates are SMT-backed verdicts for the encoded relations, not independently checked proof objects. They depend on translation, runtime capture, observable correspondence, compiled source requirements, model compilation, and the SMT backend. Unsupported translation, ambiguous correspondence, timeout, or solver unknown yields UNDECIDED. Appendix C.6 compares independent Z3 and cvc5 encodings.

5 EXPERIMENTS

Benchmarks and protocol. The experiments evaluate SMT checking effectiveness and cost, direct reconstruction, revision from counterexamples, repair baselines and host integration. A correct answer passes the benchmark or host evaluator. A model–code match means that the written mathematical model and captured executable model satisfy the exact semantic relation. An artifact match is the corresponding reference-blind check for host integrations. An output is correct and matched when its answer is correct and the applicable match holds. Problems decided is the percentage of the 3,034 assigned problems that return a decision; unavailable executable models count as unresolved. Failures repaired is the percentage of original failures that are repaired. All checks pass means a model–code match, feasible models and a verified report. Source requirements are evaluated separately.

The twelve benchmarks contain 3,034 problems. NL4Opt, NLP4LP, MAMO, OptiBench and Text2Opt provide the main formulation collections (Ramamonjison et al., 2022; AhmadiTeshnizi et al., 2024b; Huang et al., 2025b; Yang et al., 2025; Gao et al., 2026). IndustryOR, LogiOR, ComplexOR, OptMATH and BWOR extend the coverage (Huang et al., 2025a; Yang et al., 2026; Xiao et al., 2024; Lu et al., 2025; Zhang, 2026). For each of the 3,034 problems, one Qwen2.5-Instruct output at each of six sizes (1.5B–72B) yields $3{,}034 \times 6 = 18{,}204$ assigned outputs (Yang et al., 2024). References comprise 3,032 numeric objectives and two infeasibility statuses (BWOR-037 and BWOR-054) (Zhang, 2026). The controlled comparison fixes the 1,905 pre-repair refuted outputs, gives every feedback method the same Qwen2.5-14B reviser and permits one temperature-zero revision with a 32,768-token context. A numeric answer is correct when $|v - v^\star| \le 10^{-6} \max(1, |v^\star|)$.

Table 1: Complete assigned population: 3,034 problems for each of six generators, or 18,204 outputs.

Benchmark	Problems	Benchmark	Problems
MAMO easy	652	MAMO complex	211
Text2Opt template	630	OptMATH	132
OptiBench	422	IndustryOR	98
Text2Opt unstructured	248	LogiOR	76
NLP4LP	242	BWOR	75
NL4Opt	230	ComplexOR	18
Total across 12 benchmarks			**3,034**

SMT checking effectiveness. The exact queries define 1,905 refuted and 7,867 certified-equivalent original pairs. Table 2 shows that code is unavailable nearly twice as often as mathematics and contributes more unmatched restrictions.

Table 2: Executable code adds a second failure surface over 3,034 problems and six generators (18,204 outputs). The final row uses 951 fully decided pairs; categories can overlap.

Audit	Written model	Executable model
Artifact unavailable	3,202 (17.6%)	**5,980 (32.9%)**
Primary availability failure	Malformed literal, 2,174	**Runtime error, 3,723**
Redundant constraint on this side only	188	**239**
Constraint absent from the other side	415	**546**

Table 3: Baseline flags on exact relation cohorts. Rates use available decisions in each cohort.

Verifier	Refuted pairs flagged	Refutation recall (%)	Equivalent pairs flagged	False positive rate (%)
ReLoop execution	181/1,897	9.5	29/7,859	0.4
Opt-Verifier structure	1,656/1,905	86.9	4,576/7,867	58.2

Table 3 compares baseline coverage on the same exact cohorts. ReLoop flags 181 available refutations and 29 equivalent pairs; Opt-Verifier flags 1,656 and 4,576, respectively. Controlled semantic edits

and independent Z3 and cvc5 encodings appear in Appendices E and C.6. A separate source-covered audit reports source-contract certificates, refutations and undecided pairs in Appendix D.7.

SMT efficiency and scaling. Under a ten-second limit, automatic backend selection decides 94.6% of the 3,034 problems with a 0.647-second median (Table 4). On FrontierOR, exact component slicing yields paired speedups of 29.3× and 27.8× at 1,000–10,000 and 10,000–100,000 rows, respectively; all 5,055 jointly decided claims agree. Hardware details, query counts, and backend comparisons appear in Appendix C.6.

Table 4: ORSMT checking under a ten second limit on all 3,034 problems. The two settings run the same relation checks, and missing executable models are unresolved.

Setting	Problems	Decided	Problems decided (%)	Median checked time (s)
Complete audit, all problems	3,034	2,860	94.3	0.397
Automatic backend selection, all problems	3,034	2,869	94.6	0.647
Automatic selection, no captured model	134	0	0.0	–
Automatic selection, 1–4 constraints	1,508	1,508	100.0	0.550
Automatic selection, 5–24 constraints	946	942	99.6	0.694
Automatic selection, 25–99 constraints	345	343	99.4	0.892
Automatic selection, 100–999 constraints	87	72	82.8	2.592
Automatic selection, ≥1,000 constraints	14	4	28.6	10.009

Direct reconstruction. Bidirectional reconstruction raises model–code matches from 7,867 to 15,724 without language model calls by completing 6,590 programs from mathematics and 1,267 mathematical artifacts from programs. The generator analysis retains every benchmark and model size before protected selection.All 72 cells in Table 5 improve, with a median gain of 47.2 percentage points. The match rate rises from 25.0% to 93.4% at 1.5B and from 14.0% to 76.5% at 3B; repaired 1.5B outputs exceed unrepaired 72B outputs by 25.7 points.

Revision from counterexamples and ablation. Source revision raises outputs that are both correct and matched from 5,903 to 6,681 (Appendix Figure 4). On the 1,905 refuted outputs, the controlled ablation compares direct revision, plan review, row review and certified voting.

Table 5: Model–code matches before and after repair over 3,034 problems per generator (%), measured before protected selection.

Benchmark	Problems	1.5B		3B		7B		14B		32B		72B	
		Before	After	Before	After	Before	After	Before	After	Before	After	Before	After
MAMO easy	652	24.5	**99.7**	22.9	**92.6**	77.3	**100.0**	74.4	**99.7**	96.0	**100.0**	97.7	**100.0**
Text2Opt template	630	22.1	**92.7**	0.3	**52.2**	12.1	**41.7**	15.2	**52.5**	12.9	**61.6**	17.3	**63.7**
OptiBench	422	37.9	**99.8**	27.5	**90.0**	59.0	**99.1**	44.5	**95.5**	80.8	**100.0**	93.1	**100.0**
Text2Opt unstructured	248	7.7	**65.3**	4.8	**79.8**	35.5	**83.5**	56.5	**96.8**	72.6	**99.2**	68.1	**98.0**
NLP4LP	242	37.6	**99.6**	25.2	**87.2**	55.0	**99.2**	41.3	**94.2**	84.3	**100.0**	93.4	**100.0**
NL4Opt	230	36.5	**99.6**	23.9	**89.1**	53.5	**98.7**	40.4	**94.3**	80.4	**100.0**	93.9	**100.0**
MAMO complex	211	17.5	**96.7**	2.4	**78.2**	28.0	**85.3**	44.5	**90.0**	54.5	**89.1**	57.3	**85.8**
OptMATH	132	3.8	**86.4**	2.3	**28.0**	9.1	**59.8**	20.5	**71.2**	34.1	**89.4**	27.3	**87.9**
IndustryOR	98	28.6	**92.9**	11.2	**76.5**	38.8	**89.8**	38.8	**89.8**	61.2	**92.9**	55.1	**95.9**
LogiOR	76	14.5	**85.5**	7.9	**69.7**	27.6	**73.7**	44.7	**89.5**	50.0	**96.1**	43.4	**92.1**
BWOR	75	30.7	**78.7**	6.7	**66.7**	48.0	**85.3**	50.7	**94.7**	58.7	**97.3**	64.0	**94.7**
ComplexOR	18	16.7	**72.2**	5.6	**83.3**	55.6	**100.0**	72.2	**94.4**	72.2	**94.4**	66.7	**100.0**
All benchmarks	**3,034**	**25.0**	**93.4**	**14.0**	**76.5**	**44.5**	**82.1**	**44.4**	**85.6**	**63.7**	**90.3**	**67.7**	**90.3**

A separate one-round comparison gives every method the same refuted outputs and 14B reviser. Source revision followed by exact reconstruction restores an ORSMT model–code match for 1,607 outputs (84.4%); Appendix Table 26 reports the matched baseline outcomes and cost.

Comparison with other repair methods. Certified voting makes 50.9% of outputs correct and matched, 6.1 points above ReLoop and 1.4 above Opt-Verifier. The clustered interval excludes

zero only against ReLoop (Appendix Table 27); Tables 6 and 7 report the full and paired pipelines, respectively. ORSMT leads all three outcomes in Table 6; Appendix D.5 gives the paired results.

Table 6: Full-population outcomes over 3,034 problems and six generators (18,204 outputs). Outcome cells report count (rate); correct-answer counts use the 18,192 outputs with references.

Pipeline	Correct answer	Model–code match	Correct and matched	Correct and matched / correct answers	Newly correct / original correct kept/total
Original	4,931 (27.1)	7,867 (43.2)	4,588 (25.2)	4,588/4,931	0 / 4,931/4,931
ReLoop	5,760 (31.7)	8,285 (45.5)	4,744 (26.1)	4,744/5,760	829 / 4,931/4,931
Opt-Verifier	7,070 (38.9)	4,612 (25.3)	3,807 (20.9)	3,807/7,070	2,139 / 4,420/4,931
ORSMT	**7,309 (40.2)**	**16,053 (88.2)**	**7,308 (40.2)**	**7,308/7,309**	**2,378 / 4,931/4,931**

Integration with existing modeling systems. Both integrations use the same 3,034 assignments. OptiMUS-0.3 uses the exact model–code relation; OptiMind uses an exact scalar extractor or reference-blind two-of-three auditors and rechecks every reconstruction.

Table 7: Correct and matched outputs among 1,905 refuted outputs.

Policy	Correct and matched (%)
ReLoop	44.8
Opt-Verifier	49.4
ORSMT direct revision	46.4
ORSMT with plan review	47.5
ORSMT with row review	49.6
ORSMT certified voting	**50.9**

Table 8: Host integration over 3,034 problems. Cells report count (rate).

System	Pipeline	Correct answer	Artifact match	Correct and matched
OptiMUS-0.3	Original	647 (21.3)	1,113 (36.7)	630 (20.8)
	Original + ORSMT	**1,199 (39.5)**	**1,938 (63.9)**	**1,199 (39.5)**
OptiMind	Original	1,885 (62.1)	2,254 (74.3)	1,678 (55.3)
	Original + ORSMT	**1,904 (62.8)**	**2,937 (96.8)**	**1,901 (62.7)**

ORSMT adds 552 correct answers and 569 correct-and-matched outputs for OptiMUS-0.3, versus 19 and 223 for OptiMind, respectively. OptiMind also gains 683 artifact matches, showing that the integration chiefly improves artifact agreement when the host's answers are already strong. Appendix D.6 gives routes, original-failure repair rates and cost.

6 Conclusion

ORSMT closes the verification–repair loop by combining exact certificates and concrete counterexamples with bidirectional reconstruction and rechecking every adopted artifact. Across 3,034 problems and six generator sizes (18,204 outputs), it raises the model–code match rate from 43.2% to 88.2% and the share of outputs that are both correct and matched from 25.2% to 40.2%. Among 1,905 refuted outputs, certified voting yields 50.9% correct-and-matched outputs, while one source revision followed by reconstruction restores model–code agreement for 84.4% of refuted pairs. The analysis also reveals systematic asymmetries in generated artifacts: code is unavailable nearly twice as often as mathematics and is more likely to introduce additional constraints than to omit existing ones. ORSMT decides 2,869 of 3,034 problems, remains sub-second through 99 constraints, and achieves a 29.3$\times$ speedup from component slicing. Its gains on both host systems further support its use as a modular verification and repair layer for LLM-based optimization pipelines.

AI USE STATEMENT

During preparation of this manuscript, Codex (OpenAI) was used to assist with language editing and LaTeX formatting; literature discovery and bibliography checks; discussions of research ideas, experimental design, and result interpretation; and drafting and revising portions of the manuscript. It also assisted with experiment organization and consistency checks across reported results. The authors reviewed and verified the AI-assisted content and take full responsibility for the methods, experiments, numerical results, claims, citations, and conclusions. LLMs used as components of the reported method are described separately in the main text and appendix.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, Herman Brunborg, Shayan Talaei, Connor Lawless, and Madeleine Udell. OptiMUS-0.3: Using large language models to model and solve optimization problems at scale. *arXiv preprint arXiv:2407.19633*, 2024a.

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024b.

Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 415–442. Springer, 2022. doi: 10.1007/978-3-030-99524-9_24.

Yitian Chen, Cheng Cheng, Yinan Sun, Zi Ling, and Dongdong Ge. OPT-Engine: Benchmarking the limits of LLMs in optimization modeling via complexity scaling. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340. Springer, 2008. doi: 10.1007/978-3-540-78800-3_24.

Zhiqi Gao, Albert Ge, Alexander Berenbeim, Nathaniel D. Bastian, and Frederic Sala. Models can model, but can't bind: Structured grounding in text-to-optimization. In *Conference on Language Modeling*, 2026.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 73(6):2986–3009, 2025a. doi: 10.1287/opre.2024.1233.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2678–2710. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.findings-naacl.146.

Sumit Kumar Jha, Susmit Jha, Patrick Lincoln, Nathaniel D. Bastian, Alvaro Velasquez, Rickard Ewetz, and Sandeep Neema. Neuro symbolic reasoning for planning: Counterexample guided inductive synthesis using large language models and satisfiability solving. *arXiv preprint arXiv:2309.16436*, 2023.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. In *International Conference on Learning Representations*, 2025.

Minwei Kong, Chonghe Jiang, Ao Qu, Wenbin Ouyang, Zhaoming Zeng, Xiaotong Guo, Zhekai Li, Junyi Li, Yi Fan, Xinshou Zheng, et al. FrontierOR: Benchmarking LLMs' capacity for efficient algorithm design in large-scale optimization. *arXiv preprint arXiv:2605.25246*, 2026.

Haifeng Li and Mo Hai. Falsification-based verification of LLM-generated optimization models: Sound test batteries and their detection limits. *arXiv preprint arXiv:2607.16646*, 2026.

Junbo Jacob Lian, Yujun Sun, Huiling Chen, Chaoyu Zhang, Hanzhang Qin, and Chung-Piaw Teo. ReLoop: Structured modeling and behavioral verification for reliable LLM-based optimization. *arXiv preprint arXiv:2602.15983*, 2026.

Haoyang Liu, Jie Wang, Boxuan Niu, Xiongwei Han, Yian Xu, Mingxuan Ye, Zijie Geng, Fangzhou Zhu, Tao Zhong, Mingxuan Yuan, and Jianye Hao. Opt-Verifier: Unleashing the power of LLMs for optimization modeling via dual-side verification. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Theo X. Olausson, Alex Gu, Benjamin Lipkin, Cedegao E. Zhang, Armando Solar-Lezama, Joshua B. Tenenbaum, and Roger Levy. LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5153–5176. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.313.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3806–3824. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.248.

Rindra Ramamonjison, Haley Li, Timothy T. Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Augmenting operations research with auto-formulation of optimization models from problem descriptions. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 29–62. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.emnlp-industry.4.

Henry Robbins, Connor Lawless, Madeleine Udell, and Ellen Vitercik. FLARE: Verifying MILP reformulations with LLM-based theorem proving. *arXiv preprint arXiv:2608.25220*, 2026.

Zhuohan Wang, Ziwei Zhu, Ziniu Li, Congliang Chen, Yizhou Han, Yufeng Lin, Zhihang Lin, Angyang Gu, Xinglin Hu, Ruoyu Sun, and Tian Ding. ORGEval: Graph-theoretic evaluation of LLMs in optimization modeling. *arXiv preprint arXiv:2510.27610*, 2025.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-experts: When LLMs meet complex operations research problems. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HobyL1B9CZ.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Beinuo Yang, Qishen Zhou, Junyi Li, Chenxing Su, Panagiotis Angeloudis, and Simon Hu. OR-Thought: Benchmarking and automating logistics optimization modeling via structured LLM reasoning. *Artificial Intelligence for Transportation*, 6:100059, 2026. doi: 10.1016/j.ait.2026.100059.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench meets ReSocratic: Measure and improve LLMs for optimization modeling. In *International Conference on Learning Representations*, 2025.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. SATLM: Satisfiability-aided language models using declarative prompting. In *Advances in Neural Information Processing Systems*, volume 36, pp. 45548–45580, 2023.

Haotian Zhai, Connor Lawless, Ellen Vitercik, and Liu Leqi. EquivaMap: Leveraging LLMs for automatic equivalence checking of optimization formulations. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 74288–74305. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhai25a.html.

Bowen Zhang. BWOR: A bilingual benchmark for modeling and solving operations research problems from natural language. Dataset, 2026. URL https://doi.org/10.5281/zenodo.20120692.

Xinzhi Zhang, Zeyi Chen, Humishka Zope, Hugo Barbalho, Konstantina Mellou, Marco Molinaro, Janardhan Kulkarni, Ishai Menache, and Sirui Li. OptiMind: Teaching LLMs to think like optimization experts. *arXiv preprint arXiv:2509.22979*, 2025.

Yansen Zhang, Qingcan Kang, Yujie Chen, Yufei Wang, Xiongwei Han, Tao Zhong, Mingxuan Yuan, and Chen Ma. SAC-Opt: Semantic anchors for iterative correction in optimization modeling. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-sql with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 396–411. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.29.

Contents of the Appendix

A Experimental Protocol and Artifact Representations **14**

A.1 Populations, metrics and scoring 14

A.2 Artifact representations and capture 14

B Formal Guarantees and Complete Procedures **15**

B.1 Relation certificates 15

B.2 Constraint reduction and witness completion 15

B.3 Reconstruction and adoption 15

B.4 Construction and adoption procedures 16

C SMT Checking Coverage, Errors and Efficiency **17**

C.1 Checking coverage and unresolved cases 17

C.2 Input failures and model size 17

C.3 Errors introduced during code generation 19

C.4 Checking time on the controlled CPU panel 19

C.5 Canonical requirements and redundancy 21

C.6 Constraint selection and larger instances 21

D Feedback Quality, Repair Effects and Transfer **22**

D.1 Feedback coverage and error localization 22

D.2 Direct reconstruction and completion routes 23

D.3 LLM feedback, ablations and paired repair 23

D.4 LLM repair cost and recorded latency 25

D.5 Full before-and-after repair results 26

D.6 Transfer to OptiMUS-0.3 and OptiMind 27

D.7 Source requirements after repair 29

D.8 Residual constraint errors after repair 30

D.9 Results by benchmark and generator 31

E Semantic Criterion Evaluation **31**

E.1 Controlled semantic edits and preserving rewrites 31

E.2 Large edit evaluation 33

F Worked Inconsistency and Repair Cases **33**

F.1 A ReLoop pass with inconsistent feasible regions 33

F.2 Missing source requirements in two generated artifacts 35

F.3 A domain mismatch and redundant constraints 36

F.4 An incorrect Opt-Verifier judgment on a certified solution 36

The appendix follows the three experimental levels used by the paper. After fixing the populations, representations, scoring rules and formal procedures, Section C reports SMT checking coverage, unresolved cases, error distributions and running time. Section D then separates feedback quality from repair effects before reporting full before-and-after results and transfer to existing modeling systems. The repair, source-review and host-audit prompts appear beside the experiments that use them. The final sections test the evaluation criterion and give worked cases with exact witnesses.

A Experimental Protocol and Artifact Representations

A.1 Populations, metrics and scoring

The main experiment pairs all twelve benchmarks with six Qwen2.5-Instruct generators. Each problem retains its identifier, benchmark, generator, original artifacts, verifier version and adopted output. The repair cohort consists of the originally refuted pairs, and source problem resampling keeps outputs of the same problem together. Table 9 distinguishes the populations used in the main results and supplementary controls.

Table 9: Assigned populations and units used in the experiments.

Experiment	Unit	Assigned units
Main generation and repair	Problem and generator	$3{,}034 \times 6 = 18{,}204$
Paired feedback intervention	Originally refuted output	1,905
Paired constraint differences	Output with four decided comparisons	951
Controlled CPU verification	Unique problem	3,034
Canonical requirement queries	Model obligation	136,100
Canonical redundancy	Model row	340,289
Labeled edit controls	Edited or rewritten pair	1,360
Reference edit stress test	Edited pair	12,831

We report the four outcomes defined in Section 5: correct answer, model–code match, artifact match and correct and matched. A numeric answer is correct when $|z - z^\star| \leq 10^{-6} \max\{1, |z^\star|\}$. An output passes all checks when the models match, both are feasible and the report is verified. Empty feasible regions are counted separately in reconstruction outcomes. Every paired comparison preserves its assigned identifiers, including unavailable artifacts, unresolved checks and execution failures.

The original verification panel uses exact decimal arithmetic for report discrepancies. Repair scoring uses the declared tolerance for printed decisions and reference values.

A.2 Artifact representations and capture

Accepted declarations contain finitely many uniquely named real, integer or Boolean variables, optional rational bounds, linear rows with relation $\leq$, $\geq$ or equality, and an objective with its optimization direction. Structured declarations, solver objects, LP and MPS files supply these fields directly. Mathematical prose supplies a proposed transcription whose source text is retained. Malformed declarations, unsupported operators and undeclared references return a typed translation failure.

The historical panel represents each variable with a real arithmetic symbol and adds integrality predicates for integer domains. Boolean domains intersect integrality with the bounds zero and one. An absent bound remains unbounded, and each finite decimal denotes its exact rational value. Its accepted declarations require a nonempty objective term list and nonempty terms in each row. The row list itself may be empty. Region and preference comparisons use zero tolerance.

Runtime capture intercepts the optimization call, flushes pending modifications and records the instantiated model after loops and instance expansion. Later solver calls produce separate snapshots. Returned text, explicit objective and decisions, solver attributes and execution exceptions occupy separate fields. The frozen model comparison aligns all declared variables by unique normalized names, with no hidden auxiliaries. Missing or ambiguous correspondences retain an unresolved state.

Source requirements. Each obligation records its supporting source text, quantified scope, variable roles, exact coefficients, relation and applicability. Released reference formulations supply formal obligations when their adapters admit them. Open natural language supplies proposed obligations for review. Reference conventions such as symmetry constraints retain their provenance. Published objective values and status labels remain separate and enter only scoring. When an adapter admits a released formulation as a source contract, the resulting statement only checks that contract. It does not independently recover the author's intent.

Exact reconstruction. The compiler emits domains, rows, objective and direction from the selected model. A bounded product $y = bx$, where b is Boolean and $L \le x \le U$, has the exact reformulation

$$Lb \le y \le Ub, \qquad x - U(1-b) \le y \le x - L(1-b).$$

Bounded integer products use binary expansion. Added variables are existentially projected for comparison. Sequential redundancy deletion retains the model after each proved deletion, and a final relation query checks the reconstructed pair.

B Formal Guarantees and Complete Procedures

This section establishes the certificate guarantees used by verification, slicing, reconstruction and adoption. It then gives the complete procedures that instantiate those guarantees.

B.1 Relation certificates

Proof of the relation certificate. The two unsatisfiable difference queries establish both inclusions and equality of the projected feasible regions. The preference query excludes any common decision pair ordered strictly by one model and tied or oppositely ordered by the other. Their strict preference relations are equal. A satisfying query supplies the separating point or decision pair directly.

A feasible model entails a compiled source obligation when its violation formula is unsatisfiable. The returned result queries use the model, report and tolerance from Eqs. (9) and (10). Extension and value binding are existential feasibility checks. The absence of a better decision certifies optimality to the declared margin. An empty model receives its separate infeasibility outcome.

B.2 Constraint reduction and witness completion

Proof of sound reduction. The full violation formula implies the reduced formula. The reduced formula's unsatisfiability therefore implies the full formula's unsatisfiability. A complete satisfying assignment lies in the candidate feasible region and outside the obligation. A partial satisfying assignment is extended and checked against the full model before it supplies this witness.

Proof of exact component slicing. Omitted components share no variables with retained components or the obligation. Base feasibility supplies a satisfying assignment in every omitted component. Combining these assignments extends every satisfying retained assignment to the full model. The reverse implication follows by restricting a full assignment to retained variables.

Neighborhood slices smaller than the component use Eq. (11) and full witness completion. For a target with all variables observable, a directional difference decomposes into violations of its individual rows and domains. Hidden auxiliaries retain the complete projected formula. The evidence record stores the assignment, violated rows and relation used by the revision operator in Section 4.4.

B.3 Reconstruction and adoption

Proof of Proposition 1. Let T be the selected target and $\mathcal{N}(T)$ its normalized representation. Preservation of the projected region and objective graph gives $F^X_{\mathcal{N}(T)} = F^X_T$ and $G^X_{\mathcal{N}(T)} = G^X_T$. Compilation constructs the normalized tuple, and runtime capture recovers it from the generated program. Hence

$$F^X_{\widetilde{B}} = F^X_{\mathcal{N}(T)} = F^X_T, \qquad G^X_{\widetilde{B}} = G^X_{\mathcal{N}(T)} = G^X_T.$$

The exported mathematical model is $\widetilde{A} = T$. Its region and objective graph agree with those of the captured implementation. The final queries check both region inclusions and the objective relation for these constructed artifacts.

Sequential redundancy deletion preserves equivalence by induction. At each step the remaining rows entail the deleted row, and adding or removing that entailed row preserves the feasible region. The objective is unchanged. Each new deletion uses the currently retained basis.

The adoption loop accepts a candidate only after its required relation checks succeed. After an accepted construction, a failed proposal retains that certified pair and its evidence. Before such

acceptance, the return value retains the original verdicts. Source interpretation follows the compiled requirements and retained provenance of the selected target.

B.4 Construction and adoption procedures

Algorithm 1 states the complete verification, candidate generation and selection loop. Algorithms 2 and 3 expand its helper procedures. The selection order gives the mathematical artifact priority when both inputs are readable. Construction returns the mathematical model, regenerated program and computed report together. The verification procedure applies the region, objective and report queries in Section 4.2, and checks the compiled requirements enabled by the source review configuration. The policy ρ chooses direct, certified voting or protected selection without changing candidate verification.

Algorithm 1 Verification and repair

Require: Source s, compiled requirements C, artifacts $Y_0 = (A, \widehat{P}, R)$, budget K, policy ρ
Ensure: Returned artifacts Y and verification evidence e_Y

$e_0 \leftarrow \textsc{Verify}(Y_0, C), \quad \mathcal{B} \leftarrow \varnothing$
$T \leftarrow \textsc{SelectModel}(A, \widehat{P})$
if $T \neq \bot$ **then**
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{\textsc{Complete}(T, C)\}$
end if
for $k = 1, \ldots, K$ **do**
  $\widehat{P}'_k \leftarrow \mathcal{V}(s, Y_0, e_0; k), \quad T'_k \leftarrow \mathcal{H}(\widehat{P}'_k)$
  if $T'_k \neq \bot$ **then**
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{\textsc{Complete}(T'_k, C)\}$
  end if
end for
$(Y, e) \leftarrow \textsc{SelectCertified}(\mathcal{B}, Y_0, \rho)$
if $Y = \bot$ **then**
  return (Y_0, e_0)
end if
return (Y, e)

State and evidence. Each proposal Y and its evidence e form one revision record. Direct selection chooses the first fully certified record. Certified voting applies the value class quorum, and protected selection first applies its declared preservation filter. If selection returns no record, the original state and evidence remain active. The history records the selected target, triggering verdict, proposed revision and verification after construction.

Algorithm 2 Model selection and construction

procedure $\textsc{SelectModel}(A, \widehat{P})$
  if $A \neq \bot$ **then**
    $T \leftarrow A$
  else
    $T \leftarrow \mathcal{H}(\widehat{P})$
  end if
  return T
end procedure

procedure $\textsc{Complete}(T, C)$
  $\widetilde{A} \leftarrow T, \quad \widetilde{P} \leftarrow \mathrm{Compile}(\mathcal{N}(T))$
  $\widetilde{B} \leftarrow \mathcal{H}(\widetilde{P}), \quad \widetilde{R} \leftarrow \mathrm{Solve}(\widetilde{P})$
  $Y \leftarrow (\widetilde{A}, \widetilde{P}, \widetilde{R})$
  $e \leftarrow \textsc{Verify}(Y, C)$
  return (Y, e)
end procedure

Algorithm 3 Certified candidate acceptance

procedure SELECTCERTIFIED($\mathcal{B}, Y_0, \rho$)
  $E \leftarrow \{(Y, e) \in \mathcal{B} :$ every relation required by ρ is certified in $e\}$
  if ρ is protected **then**
    $E \leftarrow \{(Y, e) \in E :$ PRESERVES$(Y_0, Y, \rho)\}$
  end if
  if $E = \varnothing$ **then**
    return $(\perp, \varnothing)$
  end if
  if ρ uses certified voting **then**
    partition E by exact equality of the certified value $v(Y)$
    $G \leftarrow$ first largest value class in the fixed candidate order
    if $|G| < 2$ **then**
      return $(\perp, \varnothing)$
    end if
    return the first $(Y, e) \in G$
  end if
  return the first $(Y, e) \in E$ in fixed order
end procedure

C SMT CHECKING COVERAGE, ERRORS AND EFFICIENCY

C.1 CHECKING COVERAGE AND UNRESOLVED CASES

Table 10 separates artifact availability, model and report discrepancies, and the full formal chain. Figure 2 distinguishes the first unsuccessful stage from findings that can overlap on the same output. Redundant constraints are diagnosed separately by entailment.

Table 10: Original verification outcomes over 3,034 problems per generator, in percent.

Generator	Written model available	Executable model available	Model–code discrepancy	Report discrepancy	Any discrepancy	All checks pass
1.5B	85.6	64.3	33.2	17.4	34.0	24.2
3B	74.3	42.0	14.5	9.6	15.3	15.5
7B	72.6	75.8	9.2	8.7	12.4	45.2
14B	84.2	60.0	4.2	9.5	10.9	39.0
32B	89.2	77.3	3.8	11.8	13.1	55.8
72B	88.6	83.5	3.4	13.6	14.1	59.8

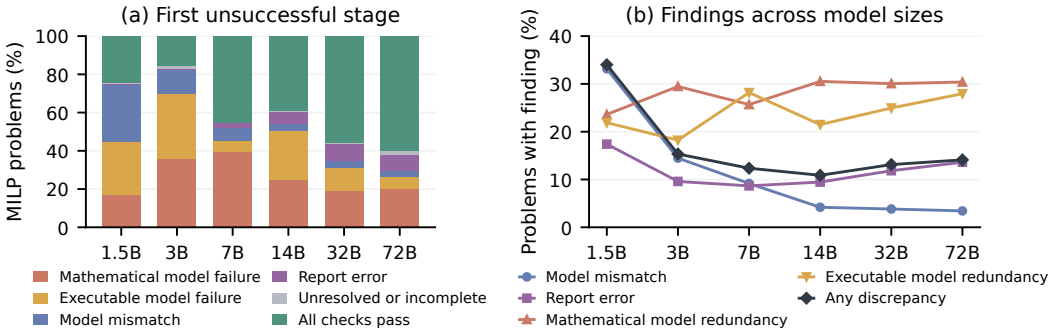


Figure 2: Verification outcomes and detected findings over 3,034 problems per generator.

C.2 INPUT FAILURES AND MODEL SIZE

Availability failures have one primary parsing or execution category, whereas semantic findings may overlap. The following tables give the input failures behind the main availability rates and the size of

the models that reach verification. Individual traces and unresolved query identifiers remain in the supplementary records.

Table 11: Written-model input failures, with 3,034 assigned problems per generator.

Written-model failure	1.5B	3B	7B	14B	32B	72B
Malformed literal	283	551	610	291	201	238
Undeclared symbol	34	82	101	117	88	91
Missing expression	21	54	31	10	29	12
Invalid comparator	29	19	61	10	0	0
Missing model fields	54	15	1	15	1	0
Missing right hand side	0	33	7	0	0	0
Duplicate variable declaration	6	4	11	4	3	0
Missing artifact	4	4	4	4	4	4
Undeclared symbols	3	1	0	20	0	0
Nonnumeric value	0	7	0	0	0	0
Unrecognized objective fields	3	1	2	1	0	0
Unrecognized constraint fields	0	5	1	0	0	0
Ambiguous term fields	0	0	0	4	1	0
Missing left hand side	0	4	0	0	0	0
Invalid operator arity	0	0	0	3	0	0
Duplicate or invalid key	0	0	1	0	0	0
Invalid variable domain	1	0	0	0	0	0
Nonliteral construct	0	1	0	0	0	0
Unbound or unsupported expression	0	0	1	0	0	0
Unrecognized variable fields	0	0	0	1	0	0
all unavailable	438	781	831	480	327	345

Table 12: Executable-model input failures, with 3,034 assigned problems per generator.

Executable-model failure	1.5B	3B	7B	14B	32B	72B
Python runtime error	811	1,224	211	865	442	170
No program in response	239	444	434	229	175	225
Python syntax error	28	86	77	110	67	93
Truncated before solve	0	1	8	7	2	8
Missing artifact	4	4	4	4	4	4
all unavailable	1,082	1,759	734	1,215	690	500

Table 13: Generated artifact size by generator, with 3,034 assigned problems per generator.

Generator	Artifact	Assigned	Available	Median variables	Median constraints	Maximum variables	Maximum constraints
1.5B	Written model	3,034	2,596	2	1	81	100
1.5B	Executable model	3,034	1,952	3	2	450	96
3B	Written model	3,034	2,253	3	3	79	130
3B	Executable model	3,034	1,275	2	3	650	1,913
7B	Written model	3,034	2,203	3	3	98	119
7B	Executable model	3,034	2,300	3	3	336	386
14B	Written model	3,034	2,554	3	4	156	134
14B	Executable model	3,034	1,819	4	4	15,150	23,396
32B	Written model	3,034	2,707	3	4	201	116
32B	Executable model	3,034	2,344	3	3	1,891	3,821
72B	Written model	3,034	2,689	3	3	128	171
72B	Executable model	3,034	2,534	3	3	5,150	10,070

Availability is counted separately for written and executable models because either artifact may be missing while the other remains usable. Medians describe the typical captured artifact, whereas the maxima expose the larger models that determine the checking limits. Table 14 then separates affine models, other admitted structures and unavailable inputs on the same assignments.

Table 14: Admitted structures and unavailable inputs over 3,034 assigned problems per generator.

	Assigned	Written model			Executable model		
Model size		Affine	Other admitted structure	Unavailable	Affine	Other admitted structure	Unavailable
1.5B	3,034	2,595	1	438	1,949	3	1,082
3B	3,034	2,249	4	781	1,265	10	1,759
7B	3,034	2,202	1	831	2,297	3	734
14B	3,034	2,551	3	480	1,811	8	1,215
32B	3,034	2,695	12	327	2,334	10	690
72B	3,034	2,689	0	345	2,527	7	500

C.3 Errors introduced during code generation

Table 15: Reusable finding taxonomy. Each label is tied to checkable evidence rather than inferred from a final objective value.

Finding	Required evidence
Additional or missing restriction	A separating assignment is feasible on one side and excluded by a named row on the other.
Domain or bound difference	A separating assignment violates a declaration while satisfying the target rows.
Objective value or preference difference	A shared feasible point has different values, or two shared points receive different orderings.
Redundant constraint	The remaining rows and fixed domains entail the row under sequential deletion.
Report error	The reported point has no feasible completion, cannot bind the reported value, or admits a better feasible point.
Transcription error	Aligned rows or declarations differ in coefficient, index, bound or direction, with source provenance.

The executable model introduces a distinct and larger failure surface even though both artifacts originate from the same model response. Across 3,034 problems and six generators (18,204 outputs), code is unavailable in 5,980 cases (32.9%), compared with 3,202 unavailable written models (17.6%). On the 951 pairs for which both sequential constraint bases are completely decided, the code adds at least one restriction not entailed by the written model in 546 cases and omits at least one written restriction in 415 cases; these categories overlap. Across the six generator sizes, redundancy appears only in code in 239 assignments and only in the written model in 188. Table 15 defines the evidence required for each reusable finding. This asymmetry motivates checking the generated implementation itself rather than treating successful execution or a matching objective value as evidence that transcription preserved the formulation.

C.4 Checking time on the controlled CPU panel

The controlled experiment selects the available executable model with the most constraints for each problem, breaking ties by variables and then generator size. It runs on two AMD EPYC 9555 processors with 128 physical cores and 755 GiB of memory, using eight concurrent problems. Each execution mode has a ten second total deadline covering input loading, preprocessing, solver processes and evidence writing. Historical evidence caches are disabled. Median checked time includes timeouts. Running both execution modes takes 541.9 seconds. Problems without a captured executable model remain unresolved in the 3,034-problem denominator.

The complete audit decides 2,860 problems with a 0.397 second median. Automatic backend selection decides 2,869 problems with a 0.647 second median. Both settings run the same ORSMT relation checks and agree on every problem that both decide. Table 16 gives automatic selection results by benchmark. Figure 3 gives the corresponding constraint size comparison.

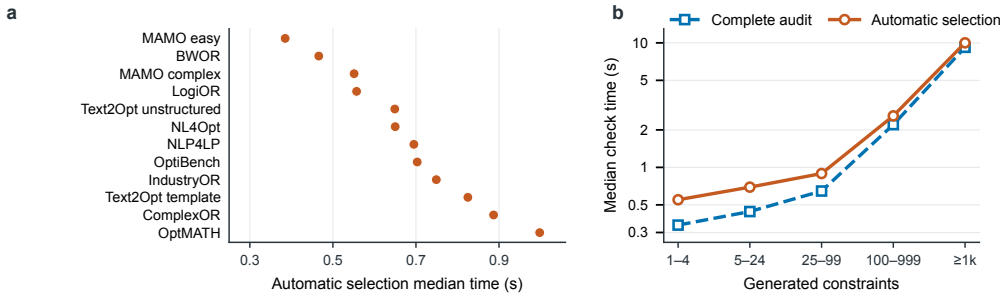


Figure 3: ORSMT checking time across (a) benchmarks with automatic backend selection and (b) constraint sizes for the complete audit and automatic selection. The panel contains all 3,034 problems. Unavailable models are unresolved, and timeouts enter the medians.

Table 16: ORSMT checking by benchmark on the 3,034-problem CPU panel.

Benchmark	Problems	Decided	Unresolved	Problems decided (%)	Median checked time (s)
MAMO easy	652	652	0	100.0	0.385
Text2Opt template	630	502	128	79.7	0.825
OptiBench	422	422	0	100.0	0.703
Text2Opt unstructured	248	228	20	91.9	0.649
NLP4LP	242	242	0	100.0	0.695
NL4Opt	230	230	0	100.0	0.650
MAMO complex	211	211	0	100.0	0.551
OptMATH	132	122	10	92.4	0.998
IndustryOR	98	95	3	96.9	0.749
LogiOR	76	73	3	96.1	0.557
BWOR	75	74	1	98.7	0.466
ComplexOR	18	18	0	100.0	0.887
All benchmarks	3,034	2,869	165	94.6	0.647

Table 17: ORSMT checking by variable domain over the complete 3,034-problem panel.

Variable domain	Problems	Decided	Problems decided (%)	Median checked time (s)
No captured model	134	0	0.0	–
Binary	119	117	98.3	0.689
Continuous	406	404	99.5	0.700
Integer	1,951	1,942	99.5	0.598
Mixed integer	424	406	95.8	0.847
All problems	3,034	2,869	94.6	0.647

C.5 CANONICAL REQUIREMENTS AND REDUNDANCY

Canonical families expose exact requirement queries and row entailment at different model structures. The timing table uses one model as its unit. The redundancy table uses one tested row and records unresolved queries separately. Sequential removal of entailed rows produces a smaller equivalent basis for subsequent comparisons and reconstruction.

Table 18: Exact requirement queries over 136,100 obligations in 1,806 canonical OR models. Times are seconds per model.

Family	Models	Obligations	Median rows	Median (s)
BinPacking	620	31,000	399	0.257
Inventory	160	12,000	75	0.013
JobShop	160	60,320	280	0.063
Knapsack	260	260	1	0.005
NetworkFlow	50	3,250	63	0.014
Pollution	80	600	8	0.015
Portfolio	160	4,640	29	0.007
Production	80	520	6	0.004
TSP	156	22,470	123	0.052
Transportation	80	1,040	13	0.009

Table 19: Exact row entailment over 340,289 rows in 1,705 canonical OR models.

Family	Checked	Rows	Redundant	Unresolved	Models with redundancy
BinPacking	575	240,761	2,336	2,612	575
Inventory	160	12,000	3,355	0	160
JobShop	155	55,320	0	2,205	0
Knapsack	260	260	0	0	0
NetworkFlow	50	3,250	1,803	0	50
Pollution	34	242	0	0	0
Portfolio	155	4,426	1,912	0	155
Production	80	520	45	0	38
TSP	156	22,470	3,586	15	156
Transportation	80	1,040	0	0	0
Total	1,705	340,289	13,037	4,832	1,134

C.6 CONSTRAINT SELECTION AND LARGER INSTANCES

The FrontierOR scaling panel (Kong et al., 2026) compares complete and restricted queries on the same instantiated models. Table 20 reports the paired timing population.

Table 20: Timing comparison on 5,869 FrontierOR claims below 100,000 constraints.

Constraints	Claims	Complete queries	Decided by both	Complete median (s)	Restricted median (s)	Paired speedup
Under 1,000	3,655	3,655	3,589	0.79	0.29	1.9
1,000 to 10,000	1,315	1,315	1,237	11.79	0.30	29.3
10,000 to 100,000	899	249	229	44.71	1.30	27.8

A reduced unsatisfiable query proves entailment. Candidate refutations are completed with a native solver and checked against every model row using tolerance $10^{-6}\max(1, |b|) + 10^{-6}$ for right hand side b. Local entailment uses exact arithmetic. The construction cap limits full queries in the largest interval. All 5,055 claims decided by both query types agree.

The extension beyond 20,000 rows deduplicates query attempts by family, instance, row and edit kind, retaining the latest record. It contains 1,383 unique claims on 78 models and excludes base

Table 21: Complete query coverage for 1,383 unique FrontierOR claims beyond 20,000 constraints.

Model rows	Models	Claims	Full query decisions	Checks decided (%)	Matching decisions
20,000 to 50,000	9	221	115	52.0	112/112
50,000 to 100,000	23	568	468	82.4	439/439
100,000 to 1 million	41	574	75	13.1	49/49
At least 1 million	5	20	0	0.0	0/0

status records. Each full model is constructed once and reused with push and pop. Capture, encoding, base feasibility and query time have separate records.

Full queries decide 658 claims, with matching verdicts on all 600 claims decided by both query types. Exact base feasibility is available for 584 claims, of which 537 full queries decide. Among 799 claims with unresolved base feasibility, 121 full queries decide. The largest recorded models reach 5,768,230 rows and 3,302,127 variables. One 372,000-row model takes 283 seconds to encode and 108 seconds to establish an exact feasible base. A weakened row query takes 5,656 seconds in the complete solver and 2.60 seconds locally. These are measurements of that individual query.

An independent solver comparison constructs obligations separately for Z3 and cvc5. It records 377,151 of 377,280 assigned query pairs, with agreement on all 110,366 pairs decided by both solvers. At least one solver remains unresolved on 266,785 recorded pairs, and 129 pairs lack a completed record. This experiment checks independent solver encodings and verdicts.

D Feedback Quality, Repair Effects and Transfer

This section separates three questions. First, what does each verifier check, and what evidence does it return? Second, how much do direct reconstruction and language-model revision repair under the same data and budget? Third, do the repaired pipelines improve correct answers, model–code matches, source requirements and host integrations without hiding residual constraint differences?

D.1 Feedback coverage and error localization

ReLoop evaluates execution behavior and selected perturbations, whereas Opt-Verifier produces structure and solution judgments (Lian et al., 2026; Liu et al., 2026). Table 22 reports their available decisions and flags on the exact model–code cohorts. Because these checks target different relations, the table keeps model–code refutations and matches in separate columns.

Table 22: Feedback coverage over 3,034 problems and six generators (18,204 outputs), with refuted and equivalent model–code pairs separated.

Verifier	Outputs with a decision / 18,204	Refuted pairs flagged / 1,905	Refuted pairs not flagged / 1,905	No decision on refuted pairs / 1,905	Equivalent pairs flagged / 7,867
ReLoop execution	16,331	181	1,716	8	29
ReLoop perturbation	803	324	51	1,530	369
Opt-Verifier structure	15,378	1,656	249	0	4,576
Opt-Verifier solution	10,202	1,633	108	164	7,001

FLARE verifies predefined formulation-family relations (Robbins et al., 2026). Its unit is therefore different from the instance-level mathematical model, executable model and report studied here. We use FLARE to distinguish the scope of the verification task; the paired feedback and repair study uses ReLoop and Opt-Verifier on the shared 1,905-output cohort.

ORSMT returns either a certificate or a separating assignment tied to rows, domains or objective preferences. Two checked cases expose errors in comparison method feedback. ReLoop accepts the IndustryOR 71 result although the mathematical and implemented feasible regions differ in both directions. Opt-Verifier rejects the certified BWOR 18 value 34 using an alternative whose listed costs sum to 37. Appendix F gives the complete models, arithmetic and repair evidence.

D.2 DIRECT RECONSTRUCTION AND COMPLETION ROUTES

Table 23 decomposes the ReLoop execution passes by overlapping semantic findings.

Table 23: ReLoop passes by discrepancy category on 1,905 refuted outputs. Categories overlap.

Discrepancy category	Affected outputs	ReLoop decisions	ReLoop passes
Additional feasible decisions	1,114	1,109	1,090
Excluded mathematical decisions	1,232	1,228	1,056
Objective preference disagreement	203	202	200
Variable declaration differences	922	915	836

Tables 24 and 25 distinguish the available source artifact and the original failure. A mathematical target reconstructs the program, and a captured program reconstructs the mathematics. This mechanical stage uses no language model calls.

Table 24: Mechanical completion routes over 3,034 problems and six generators (18,204 outputs).

Starting artifact	Assigned outputs	Model–code matches	Including empty feasible regions
Already certified pair	7,867	7,867	0
Readable written model	6,590	6,590	1,396
Captured executable model	1,267	1,267	321
Neither artifact usable	2,480	0	0

Table 25: Mechanical completion outcomes on 10,337 initially uncertified outputs.

Initial failure	Assigned outputs	Usable counterpart	Model–code matches
Original executable-model error	3,319	3,319	3,319
Missing model or implementation	2,826	839	839
Proved model inconsistency	1,905	1,905	1,905
Unsupported written representation	1,092	599	599
Unresolved relation queries	904	904	904
No shared decision variables	174	174	174
Only some variables matched	72	72	72
Unsupported runtime capture	25	25	25
Program does not invoke optimization	16	16	16
Comparison error	4	4	4

Table 5 in the main paper retains every benchmark and generator combination. Protected selection retains finite original results, reconstructs paired artifacts around those values and applies source revision to outputs without a finite result.

D.3 LLM FEEDBACK, ABLATIONS AND PAIRED REPAIR

Protected selection retains every finite original result. It reconstructs a paired artifact around that result and applies value changing revision when the original has no finite value. The mathematical artifact is the default reconstruction target, and the captured program supplies the target when the mathematical artifact is unavailable. Every route records the selected target.

The fixed 14B reviser receives the problem statement and evidence for its repair condition. All single candidate revision calls use temperature zero with a 32,768 token context. Program, model, requirement and judgment allowances are 3,072, 4,096, 8,192 and 800 tokens, respectively. Native optimization receives 30 seconds, rational reconstruction five seconds, and final checks ten seconds per query with 120 seconds per relation. The five candidate policy groups certified finite values by

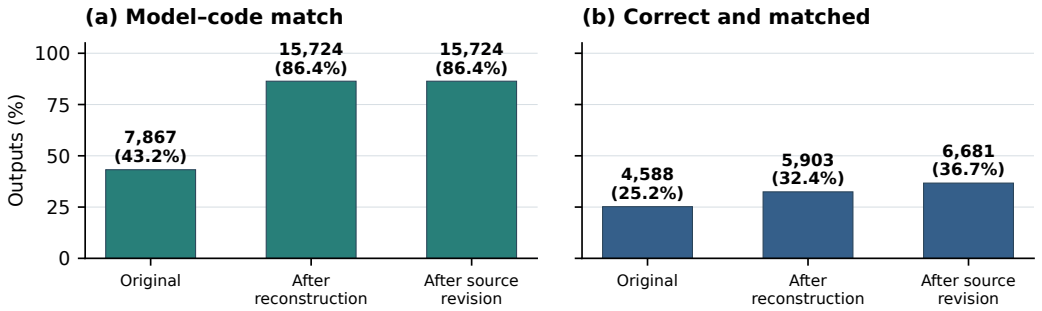


Figure 4: Model–code matches and outputs that are both correct and matched before protected selection over 3,034 problems and six generators (18,204 outputs).

exact equality, select the largest group and break ties by the fixed candidate order. Reference values enter scoring after selection.

D.3.1 Prompts used for language model revision

The matched experiment compares direct revision, which receives only a generic mismatch notice, with counterexample-guided revision, which additionally receives the failed relation and separating assignment. Both conditions use the same system instruction, problem statement, written model, previous program, reviser and decoding budget. Bracketed fields below are filled from the stored record for that output. Neither condition receives a benchmark answer or reference objective value.

Prompt for direct revision

System: You are an operations-research modeling expert.

User: You wrote a mathematical model and a gurobipy program for the problem below, and the program does not implement the model you wrote. Return corrected gurobipy code only, with no prose or Markdown fences. Change only what is wrong.

Problem: [problem statement]
Your stated model: [written model in JSON]
Your program: [previous program]
Repair instruction: A check found that the program and the model do not define the same problem.

Prompt for counterexample-guided revision

System: You are an operations-research modeling expert.

User: You wrote a mathematical model and a gurobipy program for the problem below. The written model is the implementation target. Correct only the program.

Problem: [problem statement]
Written model (JSON): [complete written model]
Previous program: [complete previous program]
Verification evidence: A checker found a concrete disagreement. [failed relations] [written-model constraints involved] [runtime-program constraints involved] [variables involved] [an assignment accepted by the program but forbidden by the model] [an assignment allowed by the model but rejected by the program] [two assignments ranked differently] [domain or bound differences]
Repair instruction: Use the counterexamples to correct the program. Return complete corrected code only.

Comparison-feedback prompt variants

Both variants use the system instruction, problem statement, written model, previous program and code-only output instruction shown above. Only the following verifier-feedback suffix changes.

ReLoop suffix: A verifier executed your program and reported: [one line per finding in the form `check (severity): message`].

Opt-Verifier suffix: A verifier read your model and your reported decision and said: `structure side:` [structure comment]; `solution side:` [solution comment].

Thus the comparison requests differ in verifier feedback rather than in the task or output format. Every returned program is recaptured and checked before it can be selected.

The implementation serializes at most 6,000 characters of the written model and 8,000 characters of the previous program. Each separating assignment is limited to 1,500 characters; the two assignments in an objective-preference witness are limited to 700 characters each. ReLoop and Opt-Verifier comments are limited to 4,000 characters. These limits and the code extractor are identical across the matched repair conditions. The extractor accepts plain code or removes one Markdown code fence, after which the complete returned program is executed, recaptured and checked.

Plan and row review start from the same source revision. Row review revises flagged candidates, adopts finite certified proposals that pass recheck, and reuses the plan condition's infeasible and unbounded branches. The compiled fallback retains previously certified pairs. Table 7 reports the voting ablation. Table 27 compares its paired outcomes with the two baselines. The interval resamples all outputs of each of the 1,531 source problems together for 10,000 draws with seed 20260913.

The one-round comparison gives every method the same refuted outputs and 14B reviser. The ORSMT arm applies one source revision request followed by exact reconstruction. The comparison measures model–code matches between the returned written and executable artifacts; Table 26 reports the outcome and generation cost.

Table 26: One-round model–code repair on 1,905 refuted outputs with Qwen2.5-14B.

Repair method	Model–code matches	Refuted pairs repaired (%)	Per output		Per matched output	
			Calls	Tokens	Calls	Tokens
ORSMT source revision and reconstruction	**1,607**	**84.4**	1.00	3,596	**1.19**	**4,263**
ReLoop	109	5.7	1.00	2,830	17.5	49,460
Opt-Verifier	68	3.6	5.80	7,949	162.5	222,690

Table 27: Clustered bootstrap comparison on 1,905 refutations for five-candidate certified voting.

Comparator	Voting only	Comparator only	Net gain percentage points	95% confidence interval percentage points
ReLoop	175	59	6.09	4.54 to 7.66
Opt-Verifier	140	113	1.42	−0.26 to 3.05

The interval is above zero against ReLoop and overlaps zero against Opt-Verifier. Together with the mechanical completion results, these comparisons separate repair without an LLM from revision driven by verifier feedback and from multi-candidate selection.

D.4 LLM REPAIR COST AND RECORDED LATENCY

Mechanical artifact reconstruction uses no language model calls. Source revision and candidate selection add generation work after that step. Table 28 reports completed records and recorded durations. Each duration measures one operation, and stages are reported separately.

Five-candidate repair contains one deterministic candidate and four samples per refutation. Its summed generation duration corresponds to 223.59 recorded seconds per intervention, before composition and verification. The 14B single-candidate mean is 56.20 seconds, and the 32B mean is 136.81

Table 28: Recorded generation and repair work over 1,905 assigned outputs per condition.

Repair setting	Candidate records	Timed records	Summed time (s)	Mean time (s)
Source revision, 14B, one candidate	1,905	1,905	107,068.3	56.20
Source revision, 14B, five candidates	9,525	9,525	425,946.3	44.72
Source revision, 32B, one candidate	1,905	1,905	260,619.2	136.81
ReLoop revision, 14B	1,905	1,903	35,219.4	18.51
Opt-Verifier revision, 14B	1,905	1,905	39,015.5	20.48

seconds on the same assigned cohort. Source-review work is tracked separately from candidate generation, and conditional branches reuse existing candidates.

D.4.1 Request and review accounting

Latency measurement. Each candidate record stores one generated program and its duration. The deterministic 14B and 32B conditions each contain one candidate per assigned output. The two additional 14B files each contain two samples per output, giving four samples beyond the deterministic candidate. Baseline records cover their respective revision operations. The duration starts and ends at the boundaries of the recorded operation. Summing it across records measures recorded work, while concurrent scheduling determines experiment wall time.

Table 29: Generation duration over 15,238 timed candidate and baseline records. Times are seconds per operation.

Operation	Timed records	Mean (s)	Median (s)
14B deterministic candidate	1,905	56.2	26.6
14B samples 1 and 2	3,810	41.9	21.2
14B samples 3 and 4	3,810	41.8	21.0
32B deterministic candidate	1,905	136.8	52.7
ReLoop 14B revision	1,903	18.5	9.0
Opt-Verifier 14B revision	1,905	20.5	10.1

The means exceed the medians in every condition, and the deterministic 32B candidate has the largest central duration on the same assigned outputs.

Candidate accounting. The five candidate policy evaluates one deterministic program and four sampled programs for each refuted output. It reconstructs each usable program, checks the resulting pair, solves the candidate model and groups eligible candidates by their certified values. The selected candidate then enters the same adoption and fallback rule as the other conditions. The deterministic candidate and the four samples account for all 9,525 candidate records. The recorded time per intervention is the summed duration divided by the 1,905 assigned outputs.

Review accounting. Source review first checks the selected mathematical interpretation. A proposed revision adds a generation operation, and a recheck evaluates the revised requirements. Both approaches share 1,452 initial checks. Plan review adds 188 revision operations and 71 rechecks, while row review adds 783 revision operations and 716 rechecks. Row review triggers more revisions because it tests individual source requirements that plan review can leave unexamined. The returned revision becomes eligible for adoption after the formal checks and the source recheck.

D.5 Full before-and-after repair results

We now move from individual repair routes to complete before-and-after pipelines. The main comparison assigns each of six generators the same 3,034 problems, for $3{,}034 \times 6 = 18{,}204$ outputs. Each method uses its own repair and adoption pipeline; ORSMT uses protected selection. ORSMT produces 7,309 correct answers, 16,053 model–code matches and 7,308 outputs satisfying both. It retains all 4,931 originally correct answers and adds 2,378. Correct answer counts and counts satisfying both use 18,192 numeric reference outputs, while model–code matches use all 18,204

assigned outputs from the 3,034 problems. Tables 30, 31 and 32 give generator results, costs and paired discordant counts.

Table 30: Correct answers by generator under protected selection over 18,192 outputs with numeric references.

Pipeline	1.5B	3B	7B	14B	32B	72B	Total
Original	213	278	801	860	1,326	1,453	4,931
ReLoop	440	682	901	912	1,348	1,477	5,760
Opt-Verifier	**1,199**	972	1,103	966	1,321	1,509	7,070
ORSMT	684	**1,037**	**1,163**	**1,378**	**1,506**	**1,541**	**7,309**

Table 31: New and retained correct answers with language model work under protected selection over 3,034 problems and six generators (18,204 outputs).

Pipeline	New correct answers	Original correct answers kept	Language model calls	Estimated tokens (millions)
Original	0	4,931/4,931	0	0.0
ReLoop	829	4,931/4,931	5,653	16.0
Opt-Verifier	2,139	4,420/4,931	81,458	120.7
ORSMT	**2,378**	**4,931/4,931**	15,249	46.1

Table 32: Paired gains and losses for protected ORSMT over 3,034 problems and six generators: 18,192 answer-scored and 18,204 model–code-checked outputs.

Metric	Comparator	ORSMT only	Comparator only
Correct answer	Original	2,378	0
Correct answer	ReLoop	1,680	131
Correct answer	Opt-Verifier	2,009	1,770
Model–code match	Original	8,186	0
Model–code match	ReLoop	7,771	3
Model–code match	Opt-Verifier	11,442	1

D.6 Transfer to OptiMUS-0.3 and OptiMind

This experiment keeps each host system's base model and 3,034 assigned problems fixed, and compares the released pipeline with the same pipeline augmented by ORSMT checking and counterexample feedback. A correct answer passes the host's released final-answer evaluator. An artifact match is the separate reference-blind audit introduced in this paper. The OptiMUS-0.3 audit applies the exact model–code relation to all 3,034 assignments. ORSMT selects 1,095 code-to-mathematics reconstructions, 533 mathematics-to-code reconstructions and 310 outputs from certified voting, with shared fallback on 1,096 assignments. Paired accounting adds 825 artifact matches and 552 correct answers. Outputs satisfying both rise from 630 to 1,199, adding 569 with no losses.

Table 33 reports the full replay over every OptiMind assignment.

ORSMT uses code-to-mathematics reconstruction, certified voting and mathematics-to-code-to-mathematics reconstruction, then retains the original output for the remaining assignments. Selection uses candidate evidence. For the original outputs, the exact extractor supports 558 scalar-formulation pairs: 283 are model–code matches and 275 remain undecided. The remaining model–code pairs are compared independently by deterministic 7B Qwen and 12B Gemma auditors with the same prompt, temperature zero, top-p one and seed zero. The prompt checks variable domains, bounds, objective, constraint families, index ranges and constants. A deterministic 32B Qwen auditor is called only when the first two disagree. Equivalent or refuted requires agreement from two of three; otherwise the pair is unresolved. The auditors do not receive benchmark answers. Each audit response has a 256-token limit and is parsed from the three requested JSON fields. Selected code-to-mathematics

Table 33: OptiMind integration on 3,034 assigned problems.

Measure	Original	Original + ORSMT
Assignments	3,034	3,034
Repaired outputs selected	0	2,902
Code-to-mathematics	0	2,837
Certified voting	0	55
Mathematics-to-code-to-mathematics	0	10
Original outputs retained	3,034	132
Artifact matches	2,254	2,937
Artifact match (%)	74.3	96.8
Correct answers	1,885	1,904
Correct answers (%)	62.1	62.8
Correct and matched	1,678	1,901
Correct and matched (%)	55.3	62.7
New artifact matches	0	+683
New correct answers	0	+19
New correct and matched outputs	0	+223
Call overhead (%)	0.0	29.9
Token overhead (%)	0.0	32.0

and mathematics-to-code-to-mathematics routes count as artifact matches only after exact recapture and relation checking; outputs from certified voting retain their ORSMT evidence.

Prompt for the reference-blind OptiMind artifact audit

System: Audit whether a written optimization model and its final Python solver program are semantically equivalent. Compare only the supplied written model and final code. Never use or infer a benchmark answer. Check decision-variable domains and bounds, objective sense and coefficients, every constraint family including index ranges, strict inequalities and constants. Ignore variable names and syntactic rewrites when a one-to-one correspondence is clear. Return JSON only with `state`, `confidence` and `reason`. Use `EQUIVALENT` only when all model components match, `REFUTED` only for a concrete mismatch and `UNRESOLVED` otherwise.

User: `WRITTEN_MODEL:` [assistant modeling text through the final code turn]
`FINAL_CODE:` [last Python code block]

Table 34 partitions correct answers and artifact matches by benchmark and confirms full population accounting. Paired accounting adds 683 artifact matches, 19 correct answers and 223 outputs satisfying both with no losses.

Table 34: OptiMind host integration counts by benchmark over 3,034 assigned problems.

Benchmark	Problems	Artifact match Original	Artifact match Original + ORSMT	Correct answer Original	Correct answer Original + ORSMT
MAMO easy	652	534	652	556	575
Text2Opt template	630	373	581	300	300
OptiBench	422	388	421	343	343
Text2Opt unstructured	248	128	231	93	93
NLP4LP	242	230	242	175	175
NL4Opt	230	222	230	209	209
MAMO complex	211	158	208	73	73
OptMATH	132	56	121	34	34
IndustryOR	98	64	93	30	30
LogiOR	76	39	71	34	34
BWOR	75	47	69	28	28
ComplexOR	18	15	18	10	10
Total	3,034	2,254	2,937	1,885	1,904

D.7 SOURCE REQUIREMENTS AFTER REPAIR

An artifact match measures agreement between the generated mathematical and executable models; source requirement checks separately measure whether that shared formulation contains the admitted requirements from the original problem. Two separate cohorts evaluate source requirements and revision. The 654-pair audit admits 4,614 typed obligations and yields 293 complete certificates, 54 explicit-source refutations, 75 symmetry-only refutations and 232 undecided pairs. Tables 35–36 follow 1,452 finite repair candidates and the 231 adopted row revisions, rerunning original and proposed programs.

D.7.1 PROMPTS FOR SOURCE REVIEW AND REVISION

Both reviews receive the original problem, the checked model and its computed optimal plan. The plan review asks for one overall requirement-by-requirement judgment. The row review makes the same task explicit for every constraint, variable declaration and objective term. A flagged output is revised from the reported findings and then reviewed again. The computed plan is evidence about the candidate model; no benchmark answer or reference objective is included.

Prompts for plan review, row review and source revision

Common system: You are an operations-research modeling expert.

Inputs to both reviews: [problem statement] [certified mathematical model] [computed objective value] [up to 200 nonzero variables in the computed optimal plan]

Plan review instruction: Check the model against the problem statement requirement by requirement: every data value and coefficient, every bound and integrality condition, every stated constraint, and the objective and its direction. Use the supplied optimal plan to test the requirements it touches. Return exactly `NO_VIOLATION` when all requirements are represented; otherwise list each misstated or omitted requirement, quoting its sentence and naming the variables involved. Do not write code.

Row review instruction: For each constraint, quote the sentence it implements and compare every coefficient and right-hand side. Check every variable type and bound, list each source requirement implemented by no row, and compare the objective coefficients and direction. Mark discrepancies as `MISMATCH` or `MISSING`, then end with `VERDICT: NO_VIOLATION` or `VERDICT: VIOLATION`. Do not write code.

Revision inputs: [problem statement] [previous program] and one of [irreducible infeasible rows and bounds], [unbounded model], or [certified model, computed plan and review findings].
Revision instruction: The problem statement is the implementation target. Correct the program so that it implements the problem statement. Return complete corrected code only.

Both reviews use temperature zero. Plan review allows 1,024 output tokens and row review allows 2,048; revision allows 3,072. The plan field contains at most 200 nonzero variables. Plan review is flagged unless its reply begins with `NO_VIOLATION`. Row review uses the final explicit `VERDICT`; if that line is absent, the presence of `MISMATCH` or `MISSING` flags the output. At most 40 flagged lines are passed to revision, with the final 1,500 characters used as a fallback when no such line is present. These parsing rules are fixed before outcome scoring.

Table 35: Source review outcomes on 1,452 finite plans before revision.

Review outcome	Plan review	Row review
Initially flagged plans	20	783
Flagged among 576 incorrect answers	11	410
Flagged among 876 correct answers	9	373
Plan revisions adopted after recheck	8	231

Table 36: Correct answers and execution outcomes on 231 adopted row review revisions.

Outcome	Original	Revised	Net change (outputs)
Correct answer from captured values	134	**174**	+40
Correct answer without exceptions	133	**171**	+38
Correct answer from explicit output	88	**118**	+30
Program exceptions	11	**4**	−7

D.8 Residual constraint errors after repair

Table 37 measures residual missing and additional constraints on the same fully decided outputs. Figure 5 gives both row totals and the direction of change per output. All three repair methods introduce constraint differences relative to the original. ORSMT introduces the fewest and leaves fewer residual differences than either baseline. These are proposals before protected adoption.

Table 37: Residual constraint differences in revision proposals before protected adoption on 951 fully decided outputs.

Pipeline	Outputs with missing constraints	Outputs with additional constraints	Outputs without constraint differences	Constraint precision (%)	Constraint recall (%)
Original	415	546	256	72.0	78.6
ReLoop	523	662	176	60.6	73.6
Opt-Verifier	544	673	159	58.6	71.3
ORSMT	**462**	**614**	**227**	**66.4**	**77.0**

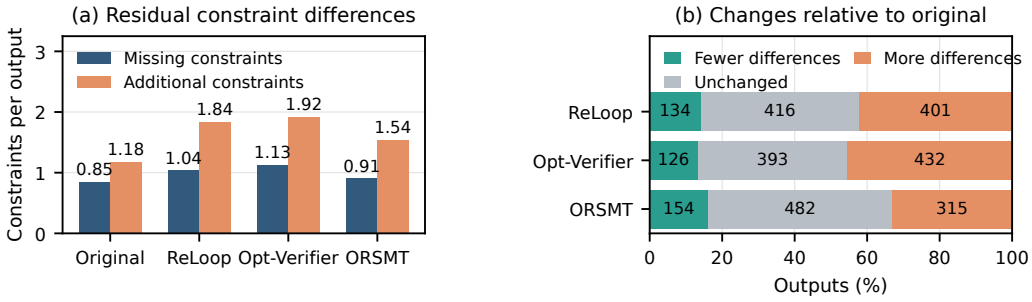


Figure 5: Constraint differences on 951 paired outputs. Panel (b) labels give output counts.

The paired row comparison holds the original mathematical artifact fixed. Sequential entailment removes redundant rows on each side. Missing constraints are mathematical basis rows not entailed by the program, and additional constraints are program basis rows not entailed by the mathematics. Constraint recall and precision divide the entailed rows by their respective basis sizes. The complete comparison requires every relevant row query to decide. Undefined denominators remain outside conditional precision and recall rates.

The experiment checks 7,620 original and revised pairs with exact rational arithmetic and integer domains, allowing 200 rows per artifact, five seconds per row and 90 seconds per pair. It uses 12 CPU workers and finishes in 66.0 seconds without language model calls. Eight original records with unresolved row queries are excluded from complete decisions. The shared intersection contains 951 outputs. Table 38 connects residual differences to apparently correct values.

Appendix F expands the formal counterexamples and repair outcomes. The cases include missed region differences, shared omissions, domain errors, redundant rows and an incorrect negative solution judgment.

Table 38: Correct answers and changed constraints on the same 951 fully decided outputs.

Method	Correct answers	Correct answers with constraint differences	Correct answers with more differences	New differences among 256 originals without differences
ORSMT	556	456	215	116
ReLoop	605	505	260	150
Opt-Verifier	661	556	303	161

D.9 Results by benchmark and generator

Table 39 reports outputs satisfying both conditions for every benchmark and generator under protected selection. Each cell requires a correct answer and a model–code match. The problem column gives the denominator for each generator. These counts resolve the benchmark and generator summaries in the main text and sum to 7,308 outputs satisfying both without averaging benchmark percentages. Small generators gain strongly on the formulation benchmarks, while the lower counts on the two Text2Opt families and OptMATH identify benchmarks where fewer outputs satisfy both.

Table 39: Outputs with a correct answer and model–code match by benchmark and generator after protected selection, over 18,192 outputs with numeric references.

Benchmark	Problems	Stage	1.5B	3B	7B	14B	32B	72B
MAMO easy	652	Original	32	81	388	422	580	570
		ORSMT	271	396	485	551	596	575
Text2Opt template	630	Original	0	0	6	32	15	79
		ORSMT	111	55	45	80	82	110
OptiBench	422	Original	24	55	142	139	253	277
		ORSMT	117	218	236	279	299	290
Text2Opt unstructured	248	Original	0	2	16	61	91	92
		ORSMT	15	17	22	64	93	98
NLP4LP	242	Original	13	20	64	65	138	143
		ORSMT	68	109	125	144	158	156
NL4Opt	230	Original	6	20	80	76	150	165
		ORSMT	60	141	153	169	181	176
MAMO complex	211	Original	0	2	29	29	9	53
		ORSMT	16	62	57	44	17	63
OptMATH	132	Original	1	0	1	2	10	7
		ORSMT	8	9	6	8	18	13
IndustryOR	98	Original	0	1	10	7	26	20
		ORSMT	7	13	18	12	27	22
LogiOR	76	Original	0	0	1	7	9	11
		ORSMT	3	6	4	12	12	14
BWOR	75	Original	1	0	6	5	11	14
		ORSMT	4	4	7	9	14	15
ComplexOR	18	Original	0	0	1	5	7	6
		ORSMT	3	7	5	6	9	9

E Semantic Criterion Evaluation

E.1 Controlled semantic edits and preserving rewrites

Table 40 applies five semantic edit operators and five preserving rewrites.

Exact region and preference queries label the resulting model relations. Of 640 applicable edits, 562 are different, 67 are equivalent and 11 remain unresolved. Each of the five preserving operators contributes 144 equivalent pairs. Outcome value compares the optimized value, mapped optimum compares optimizing decisions after variable mapping, structural matching compares model graphs, and falsification samples separating points. Table 41 reports unresolved outcomes instead of dropping them. These controls evaluate artifact semantics under the formal reference protocol.

Table 40: Controlled semantic edits and preserving rewrites over 1,360 artifact pairs.

Edit operator	Applicable	Proved different	Proved equivalent	Unresolved
Drop constraint	144	123	19	2
Tighten right hand side	130	121	9	0
Flip relation	129	128	0	1
Relax integrality	94	68	20	6
Perturb coefficient	143	122	19	2
All edits	640	562	67	11

Preserving rewrite	Pairs	Certified equivalent	Refuted or unresolved
Scale one constraint	144	144	0
Duplicate one constraint	144	144	0
Reorder variables and constraints	144	144	0
Rename variables	144	144	0
Positively scale the objective	144	144	0
All preserving rewrites	720	720	0

Table 41: Criterion decisions over 562 proved different and 787 proved equivalent model pairs.

	Proved different			Proved equivalent		
Criterion	Detected	Missed	Unresolved	Accepted	Falsely flagged	Unresolved
Outcome value	313	197	52	613	138	36
Mapped optimum	358	152	52	569	182	36
Structural graph	562	0	0	288	499	0
Falsification with one point	286	276	0	785	2	0
Falsification with four points	427	135	0	783	4	0

Table 42: Rates and 95% Wilson intervals on 562 different and 787 equivalent pairs.

Criterion	Changed pairs detected (%) with 95% interval	Equivalent pairs accepted (%) with 95% interval	Pairs decided (%)
Outcome value	55.7 [51.6, 59.7]	77.9 [74.9, 80.7]	93.5
Mapped optimum	63.7 [59.6, 67.6]	72.3 [69.1, 75.3]	93.5
Structural graph	100.0 [99.3, 100.0]	36.6 [33.3, 40.0]	100.0
Falsification with one point	50.9 [46.8, 55.0]	99.7 [99.1, 99.9]	100.0
Falsification with four points	76.0 [72.3, 79.3]	99.5 [98.7, 99.8]	100.0
Exact reference relation	100.0 [99.3, 100.0]	100.0 [99.5, 100.0]	100.0

Four-point falsification gives the best balance between detecting changed pairs and accepting equivalent pairs. Using the exact semantic relation as the reference partition, we compare weaker evaluation criteria on the proved model pairs.

E.2 Large edit evaluation

The larger panel tests which model changes remain invisible to objective values or to a single region inclusion. Table 43 combines the exact partition and the corresponding miss rates by operator. Outcome value misses 4,721 proved changes and also falsely separates 13 preserving edits. A test in a single inclusion direction misses 5,738 proved changes. Constraint deletion and integrality relaxation often preserve the optimum, while tightening requires the opposite inclusion direction to expose excluded decisions.

Table 43: Partitions and miss rates among 12,831 edited pairs. Rates use proved changes in each row.

Edit operator	Applicable	Proved different	Proved equivalent	Unresolved	Value test miss rate (%)	One direction miss rate (%)
Drop constraint	2,926	2,619	298	9	53.5	0.0
Tighten right hand side	2,489	2,330	153	6	36.5	100.0
Flip relation	2,463	2,420	33	10	13.2	96.6
Relax integrality	2,028	1,240	721	67	63.6	0.0
Perturb coefficient	2,925	2,172	743	10	62.7	49.3
All edits	12,831	10,781	1,948	102	43.8	53.2

F Worked Inconsistency and Repair Cases

The following four recorded benchmark cases connect the original artifacts, verification evidence, repair and subsequent checks. They connect individual mechanisms to exact variable domains, objective directions and constraint coefficients.

F.1 A ReLoop pass with inconsistent feasible regions

IndustryOR 71: a ReLoop pass with opposite model and code constraints

Mathematical model. The 14 billion parameter generator returns four nonnegative integer quantities $\mathrm{boat}, \mathrm{plane}, \mathrm{train}, \mathrm{truck}$. Both artifacts use the same objective and resource limits.

Written restriction.

$$\mathrm{plane} - \mathrm{boat} \geq 0.$$

Solver code.

```
m.addConstr(boat - plane >= 0)
```

Returned result. The reported objective is 623, equal to the benchmark reference. ReLoop returns VERIFIED with confidence 1.

SMT counterexamples. In the decision order shown at left, $(0, 1, 0, 0)$ satisfies every written row but violates the executed restriction. The point $(1, 0, 1, 0)$ satisfies every executed row but violates the written restriction. Both region inclusions are REFUTED.

Mathematics as the target. Regenerate the program with the written restriction.

Code as the target. Recover the mathematical model with the executed restriction.

Recheck. Each reconstructed pair has identical objectives and two UNSAT region difference queries.

Figure 6: A ReLoop pass with exact counterexamples and two verified reconstruction directions.

IndustryOR problem 71 at 14B returns the reference value 623. The mathematical model and program each contain four variables and five constraints. ReLoop accepts the program's execution, and the reported solution passes feasibility, value binding and optimality checks under both models. The models nevertheless permit different decisions. Figure 6 shows the original judgment, the two witnesses and the two repair directions.

Complete compared models. Let b, p, r, t denote boat, plane, train and truck quantities. All four are nonnegative integers. Both artifacts maximize

$$5t + 10p + 8b + 7r \tag{12}$$

and contain the four common constraints

$$\begin{aligned} 12t + 20p + 15b + 10r &\leq 890, \\ 6t + 3p + 5b + 4r &\leq 500, \\ t - r &\leq 0, \\ b - r &\leq 0. \end{aligned} \tag{13}$$

The remaining mathematical row is $p - b \geq 0$, while the program adds $b - p \geq 0$. Their objective coefficients, row counts and variable correspondence agree. The different inequality restricts the feasible region in both directions.

Witness checks. The decision $(b, p, r, t) = (0, 1, 0, 0)$ satisfies every mathematical constraint. Its wood and steel use are 20 and three, and its executable boat row has value negative one. Conversely, $(1, 0, 1, 0)$ satisfies the program's complete model, uses 25 units of wood and nine of steel, and violates the written boat row by one. These decisions are witnesses to the two set differences, independently of the optimum shared by the models.

Table 44: Exact witness evaluations for IndustryOR 71 at 14B.

Quantity or check	Mathematical witness	Executable witness
Decision (b, p, r, t)	$(0, 1, 0, 0)$	$(1, 0, 1, 0)$
Wood use, limit 890	20	25
Steel use, limit 500	3	9
Mathematical row $p - b$, lower bound 0	1	-1
Executable row $b - p$, lower bound 0	-1	1
Objective value	10	15
All mathematical constraints satisfied	Yes	No
All executable constraints satisfied	No	Yes

Repair from the mathematical model. Keeping the mathematical model as the selected target replaces the executable inequality with $p - b \geq 0$. The compiler also reproduces the four common rows, integer declarations and objective. Capturing the new program yields the same complete model. Both difference queries are unsatisfiable, and the objective expressions are equal. This direction repairs the inconsistent implementation from the available mathematics.

Repair from the program. Keeping the captured program as the selected target exports $b - p \geq 0$ into the mathematical artifact and retains every other captured field. The regenerated program and exported model again have unsatisfiable difference queries and equal objectives. This direction completes or corrects mathematics from an executable model. The source requirements determine which selected interpretation should be retained.

IndustryOR 71: the changed code and the certified result

Original program row.

```
m.addConstr(boat - plane >= 0)
```

Mathematical target repair.

```
m.addConstr(plane - boat >= 0)
```

The variable declarations, objective and four other rows are retained. The program prints its computed objective and variable assignments after optimization.

Certification after construction.

Both region difference queries are unsatisfiable. The objective coefficients and direction match. A decision with value 623 exists, and the query for a value greater than 623 is unsatisfiable.

Effect of the repair.

The model–code relation changes from refuted to certified even though the answer remains correct.

Executed reconstruction. A fresh replay executes the original program and both reconstructed Gurobi programs. It reads the instantiated variables, bounds, rows and objective from each program and compares them with the selected target using exact Z3 queries. Both repaired programs return $(b, p, r, t) = (0, 0, 89, 0)$ with value 623. The returned point satisfies the complete model and binds to

that value, and no better feasible decision exists. This replay checks the actual constructed programs in addition to the displayed row calculations.

F.2 Missing source requirements in two generated artifacts

MAMO easy problem 20 at 1.5B illustrates two distinct repairs. The first restores agreement between the generated mathematical model and code. The second restores source requirements that both artifacts omitted and changes the optimal result.

Source and generated models. The problem chooses integer quantities of four tour packages, with costs 100, 200, 300 and 400. It gives two capacity limits and two minimum quality requirements. The generated model uses nonnegative integers x_1, x_2, x_3, x_4, minimizes

$$f(x) = 100x_1 + 200x_2 + 300x_3 + 400x_4, \tag{14}$$

and contains only $x_1 + x_2 \leq 50$. The program also includes $x_3 + x_4 \leq 80$. Both artifacts omit the quality requirements. Their reported decision is zero in every coordinate, with objective zero. The recorded report has no explicit solver status.

MAMO easy 20: disagreement and missing requirements

Mathematical artifact.
$x_i \in \mathbb{Z}_{\geq 0}$, minimize $f(x)$,
$x_1 + x_2 \leq 50$.

Additional code row.
$x_3 + x_4 \leq 80$.

Exact counterexample.
$(0, 0, 81, 0)$ satisfies the written model and violates the additional capacity row by one.

Omitted source requirements.
$0.6x_1 + 0.8x_2 \geq 20$,
$0.7x_3 + 0.9x_4 \geq 30$.

Old report.
$(0, 0, 0, 0)$ has zero cost and is optimal under both incomplete artifacts, while violating both source quality requirements.

Artifact repair. Adding the second capacity row to the written model makes its region identical to the program's region. The objective and domains were equal before this change and remain equal. The zero decision remains optimal. This repair resolves the artifact discrepancy without changing the interpretation shared by both generated models.

Source repair. The reviewed source formulation contains both capacities and both quality requirements. Clearing the decimal denominators gives the complete repaired model

$$\begin{aligned} \min \quad & 100x_1 + 200x_2 + 300x_3 + 400x_4 \\ \text{subject to} \quad & x_1 + x_2 \leq 50, \qquad x_3 + x_4 \leq 80, \\ & 3x_1 + 4x_2 \geq 100, \qquad 7x_3 + 9x_4 \geq 300, \\ & x_1, x_2, x_3, x_4 \in \mathbb{Z}_{\geq 0}. \end{aligned} \tag{15}$$

The integer coefficients represent the source decimals exactly. The old zero report fails both quality checks. The repaired decision $(34, 0, 43, 0)$ has value 16,300 and matches the released answer. A fresh exact query for a feasible decision with lower cost is unsatisfiable.

Table 45: Feasibility and value checks for the repaired MAMO easy 20 decision.

Constraint or objective	Evaluated value	Required relation
First capacity, $x_1 + x_2$	34	At most 50
Second capacity, $x_3 + x_4$	43	At most 80
First quality, $3x_1 + 4x_2$	102	At least 100
Second quality, $7x_3 + 9x_4$	301	At least 300
Objective, $f(x)$	16,300	Equals reported value
Feasible decision with smaller objective	None	Query unsatisfiable

Result of each intervention. Model–code verification first restores the missing capacity row, changing the mathematical/executable row counts from one/two to two/two and the model–code relation from refuted to certified; the objective remains zero and the answer remains incorrect. Source

review then restores both quality rows in both artifacts, giving four/four rows, objective 16,300 and a correct answer. Code construction propagates the completed mathematical model into the implementation; the returned decision passes the feasibility and value checks above.

F.3 A DOMAIN MISMATCH AND REDUNDANT CONSTRAINTS

ComplexOR problem 3 at 1.5B shows why row agreement alone is insufficient. The written model uses nonnegative integers x, y, maximizes $x + y$, and repeats the same row three times:

$$c_1 : x + y \leq 2, \qquad c_2 : x + y \leq 2, \qquad c_3 : x + y \leq 2. \tag{16}$$

The code uses the same objective and rows, but declares x binary. The decision $(2, 0)$ satisfies all rows of both artifacts and violates only the executable domain. The original report $(0, 2)$, with value two, is feasible, correctly valued and optimal under both models.

ComplexOR 3: domain repair and a smaller equivalent constraint set

Domain difference.
Mathematics: $x \in \mathbb{Z}_{\geq 0}$.
Code: $x \in \{0, 1\}$.

Separating decision.
At $(x, y) = (2, 0)$, each row has left hand side two, while the binary domain rejects $x = 2$.

Domain repair.
Declare the code variable as a nonnegative integer and check both inclusions again.

Redundancy evidence.
For each duplicate, the other two rows entail it. Delete c_3, then test and delete c_2 against the remaining model.

Final model.
One row $c_1 : x + y \leq 2$ remains. The integer domains and objective are unchanged. The maximum value remains two.

Checked stages. The original three-row mathematical model is the target, whereas the three-row executable model is refuted because it makes x binary. Restoring x as a nonnegative integer certifies the pair. Sequentially removing one and then two duplicate rows retains certification, leaving one essential row with the same domains and objective.

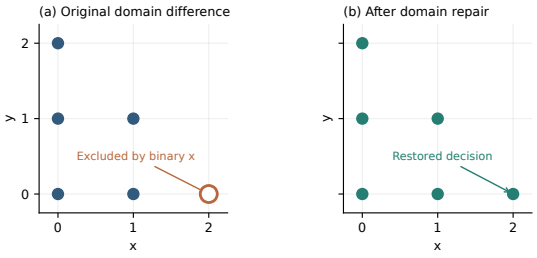


Figure 7: ComplexOR 3: the original binary domain excludes the circled integer decision; repair restores it.

Sequential deletion. To remove c_3, the checker proves that c_1, c_2 and $x + y > 2$ cannot hold together. After deleting c_3, the same query with c_1 proves c_2 redundant. The remaining row is essential because $(3, 0)$ violates the original bound. Thus each deletion preserves an equivalent constraint set.

Source interpretation. The source concerns three participants, three cars and an interest matrix, whereas both artifacts use two variables. Domain and redundancy repair certifies the generated pair, but restoring the source requires indexed assignment variables and constraints. The released answer is three; the repaired generated model still has optimum two.

F.4 AN INCORRECT OPT-VERIFIER JUDGMENT ON A CERTIFIED SOLUTION

BWOR 18: a correct result receives incorrect revision advice

Returned assignment.
The companies choose stores 3, 2, 1, 4 and 5. The costs are 7, 9, 6, 6 and 6, giving 34 in total.

Formal checks.
The model pair is consistent. The decision is feasible, its value is bound correctly, and no better feasible decision exists in the compared models.

Verifier comment.
The proposed alternative uses stores 1, 2, 3, 4 and 5. Its listed costs are 4, 9, 12, 6 and 6. The comment labels this total as 34.

Arithmetic check.
The listed alternative costs total 37. The alternative supplies no improvement over the certified value 34.

BWOR problem 18 at 14B assigns five construction companies to five stores. Its recorded model contains 25 variables and ten constraints. Both region inclusions, objective coefficients and preferences agree. The reported objective 34 matches the reference value, and the recorded exact checks certify both region inclusions, equal objective coefficients, equal decision preferences, feasibility under both models and objective binding. The query for a better feasible decision is unsatisfiable.

The Opt-Verifier reproduction accepts the structural comparison but returns a negative solution judgment. Its proposed alternative has store indices $(1, 2, 3, 4, 5)$ and costs $(4, 9, 12, 6, 6)$, which sum to 37 rather than the claimed 34. The returned assignment $(3, 2, 1, 4, 5)$ instead costs (7+9+6+6+6=34). The error is in the feedback arithmetic, not model extraction or variable alignment.

Exact decision path and consequence. The checker proves both region inclusions and equal objective coefficients, substitutes the returned assignment into every row, binds the reported value to the evaluated objective and proves that no feasible assignment has value below 34. The archived comparison judgment uses five language model calls and 44.2 seconds, yet its proposed counterexample is strictly worse. The exact certificate therefore retains the correct program and value instead of triggering an unnecessary revision.