VERIFIER-NATIVE CONTROL AND TRAJECTORY EVALUATION FOR OPERATIONS RESEARCH REPAIR AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Operations Research (OR) repair is iterative: a candidate edit is solved, checked, diagnosed, and revised. We study how LLM agents use this feedback while preserving operational validity. Repair-Sufficient State (RSS) compiles solver and domain-checker observations into typed failure state; Counterfactual Verifier Loop (CVL) organizes probes and repairs around verifier transitions; and OR-INSPECT-BENCH evaluates fixed-trace diagnosis and live repair. Across 19 directly measured model snapshots, the full pipeline improves strict recovery over one-shot prompting. Matched controls show that RSS improves on raw multi-turn feedback across three anchors. In a fresh same-window Qwen comparison, adding reference-free output normalization to RSS raises strict all-error recovery from 30.6% to 42.8%; the full CVL-r2 configuration reaches 45.8%, a further 3.1-point estimate. A Qwen factorial also finds that canonical-emission correction raises strict recovery by 14.4 and 20.6 points with probes enabled and disabled. Across six paired model anchors, additional control gains vary by model. OR-INSPECT-BENCH combines fixed-trace diagnosis, live repair, and composition holdouts. Together, the results quantify contributions from typed verifier state and output handling, and characterize when added control improves recovery.

1 INTRODUCTION

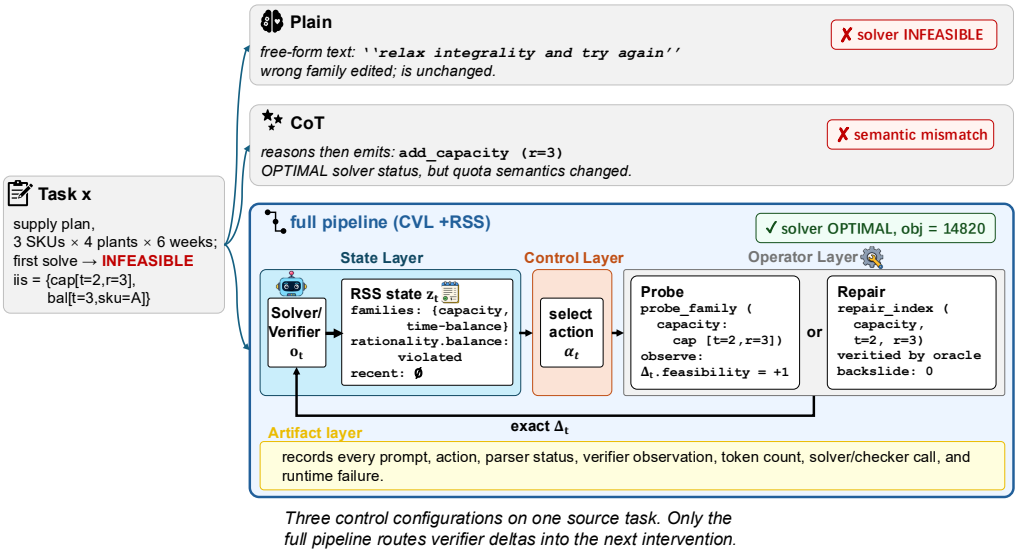
Optimization models are rarely written correctly in a single pass. An OR formulation defines decision variables, an objective, and constraints. A proposed formulation is solved, the solver exposes infeasibility or rationality evidence, and the modeler decides which constraint, index, domain rule, or objective term to repair. Evaluating LLM agents in this setting requires a protocol and a controller for the repair process itself.

Most OR-LLM benchmarks focus on the endpoint formulation. Examples include NL4Opt, MAMO, LEAN-LLM-OPT, and ORLM (Ramamonjison et al., 2023; Huang et al., 2025b; Liang et al., 2026; Huang et al., 2025a). A model reads a problem statement, emits a formulation, solves once, and is graded on the answer. This protocol is valuable for translation. Repair evaluation adds the process view: whether the agent can use solver/domain evidence over a sequence of targeted interventions.

Consider a supply-planning model that returns `INFEASIBLE` with a conflict over capacity and time-indexed balance. An irreducible infeasible subsystem (IIS) identifies an inclusion-minimal conflicting constraint set. An OR practitioner reads the conflict, hypothesizes which constraint family is responsible, probes the verifier state, and then commits to a repair. Feasibility alone is still insufficient: the repair can change quota semantics, relax integrality, or optimize the wrong objective. The relevant capability is trajectory-level decision-making under exact verifier feedback, where the agent improves one signal without damaging OR semantics already satisfied.

Verifier-mediated OR repair makes the information available at each decision explicit: solver status, conflict structure, feasibility margins, domain rationality checks, and action history. RSS compiles these signals into a typed five-channel state; CVL uses verifier transitions to configure inference-time control; and OR-INSPECT-BENCH evaluates both fixed-trace interpretation and live repair. Our experimental design isolates a distinct interface-level question: how typed feedback, output handling, and controller configuration affect repair on matched cases, and how these component effects relate to trajectory-level diagnosis under a shared evaluation protocol (Appendix F).

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.



Three control configurations on one source task. Only the full pipeline routes verifier deltas into the next intervention.

Figure 1: Illustration of verifier-guided repair with the full controller. On the same infeasible supply-planning instance, plain prompting has only the task statement, CoT adds a self-generated rationale, and CVL+RSS uses exact verifier-state deltas as the control signal. The verifier-induced state transition guides the next probe or repair.

The empirical question is which parts of this interface change repair outcomes. The system matrix contains 19 directly measured model snapshots and four explicitly marked imputed reference rows; among the measured snapshots, the full pipeline improves on source-matched one-shot prompting for most models (Fig. 4). Matched controls show that typed RSS improves on raw multi-turn feedback across three anchors. A new same-window Qwen comparison directly adds the reference-free output normalizer to RSS: strict all-error recovery rises from 110/360 to 154/360, while the full CVL-r2 configuration reaches 165/360. A separate Qwen probe-by-canonicalization factorial identifies gains from canonical-emission correction under both probe settings. A complementary TRACEJUDGE lane evaluates interpretation of the same verifier evidence in completed trajectories.

Our contributions are:

- **Typed verifier state for OR repair.** RSS turns solver status, conflict structure, rationality checks, action history, and parser outcomes into a five-channel interface; matched experiments quantify its benefit over untyped multi-turn feedback.
- **Component-resolved inference-time control.** We compare raw RSS, RSS with reference-free output normalization, and CVL configurations on matched Qwen cases, and test canonical-emission correction under two probe settings. These interventions identify a positive normalization effect and quantify the additional margin of the full control package.
- **Paired trajectory evaluation.** OR-INSPECT-BENCH joins fixed-trace diagnosis and live repair under shared verifier-state concepts, with strict scoring, acceptance audits, and source-task-aware comparisons.

Together, these contributions make verifier information, control choices, and repair outcomes directly comparable while preserving their distinct roles in an inference-time OR repair system.

2 VERIFIER-NATIVE REPAIR INTERFACE

The agent acts on solver/domain feedback, and each intervention produces a verifier state and an execution record. Fig. 1 illustrates the full-probe path on a supply-planning instance.

2.1 SYSTEM OVERVIEW

The system separates four responsibilities. RSS converts raw solver and domain feedback into typed repair state. CVL chooses diagnostic probes and repairs from that state. The operator memory retrieves reusable local actions and ranks them under the verifier budget. The execution record stores prompts, parsed actions, verifier observations, token counts, solver/checker calls, parser status, and runtime failures. Recorded calls and tokens distinguish the cost of interaction from repair quality; provider failures are tracked separately from malformed model outputs.

2.2 REPAIR-SUFFICIENT STATE (RSS)

For task x, verifier observation o_t, and history h_t, the RSS compiler emits

$$z_t = \mathcal{C}(x, o_t, h_t). \tag{1}$$

The state preserves five channels: solver state, conflicts (including constraint-family labels), rationality checks, intervention history, and parser state. For the running supply-planning instance, these channels record `INFEASIBLE`, the two conflicting families `{capacity, time-balance}`, the violated balance check, and the failed intervention `relax_integrality`. RSS keeps the compact verifier-grounded information needed to choose the next local intervention.

2.3 COUNTERFACTUAL VERIFIER LOOP (CVL)

At state z_t, the agent proposes a structured probe or repair. The adapter executes the candidate on a copied state and returns verifier evidence. For a parent snapshot b and child a, the finite transition features are

$$\Delta_t = (\Delta_{\text{feas}}, \Delta_{\text{rat}}), \quad \Delta_{\text{feas}} = \frac{\text{IISgap}(b) - \text{IISgap}(a)}{\max(1, \text{IISgap}(b))}, \quad \Delta_{\text{rat}} = \text{rat}(a) - \text{rat}(b), \tag{2}$$

where IISgap and rat are the adapter's infeasibility and rationality diagnostics. A scalar transition reward backs up these observations through the search tree. UCT governs traversal; a deterministic lexicographic key governs greedy preference and fallback extraction. Appendix A separates these stages, including tie rules and termination, rather than treating a state projection as the entire action-selection rule.

A typed action declares an operator, a target `<family>::<symbol>`, a canonical `after` value, and an expected-delta field. The full CVL configuration permits diagnostic probes, such as testing a constraint family, as well as repairs. The measured transition, rather than a model's verbal self-assessment, supplies the search signal. This relates to tool-grounded reflection and search (Shinn et al., 2023; Gou et al., 2024; Zhou et al., 2024; Zhang et al., 2026a), while using an OR adapter instead of training a process-reward model (Wang et al., 2024; Lightman et al., 2024; Luo et al., 2024).

Configuration for matched comparisons. In these comparisons, CVL disables diagnostic probes while retaining verifier-guided candidate selection. It also normalizes schema aliases and completes a typed constraint reference when its emitted field and target agree. An omitted `after` value may be filled using the visible corrupted artifact or public schema; an explicit value is never overwritten. This is a model-facing repair aid, not a change to the benchmark scorer. Its comparison with RSS consequently measures the combined selection-and-normalization configuration, not ranking alone. Appendix A.3 specifies this boundary.

2.4 OPERATORS, MEMORY, AND SEARCH

Repair operators cover objective direction, integrality, time indices, capacity, and scenario probabilities. Retrieval uses RSS signatures rather than source identifiers, oracle labels, or reference answers. The matched controller study uses run-isolated experience stores with online writes disabled; it is distinct from the broader full-pipeline study. We retain both configurations and report within-study comparisons rather than subtracting scores across evaluation windows. Candidate-generation caps, prompts, model snapshots, parser versions, and recorded calls are part of the run specification.

Table 1: Evaluation lanes. The public lane preserves each benchmark's native modeling-accuracy (MA) metric, while the two OR-INSPECT-BENCH lanes focus on fixed-trace diagnosis and live verifier-guided repair.

Lane	Scored object	Capability measured
External Transfer	Native MA score	Transfer to established public OR-modeling protocols
Trace Inspection	Composite diagnostic score	Recognition of decisive failure evidence and the next verifier action
Live Repair	Strict and semantic repair rate	Verifier-guided improvement under active multi-turn repair

2.5 RUN RECORDS

The same run record links the live run to the reported tables. Each row stores the source task, model/provider, prompts, parsed actions, verifier outputs, tokens, calls, final status, failure category, and supersession metadata. These records support the aggregate tables, identify rows that require rerun, and keep model capability separate from API reliability (§3.4, Appendix D).

3 OR-INSPECT-BENCH: A TRAJECTORY-AWARE OR REPAIR BENCHMARK

OR-INSPECT-BENCH evaluates verifier-rich OR repair through typed traces and a shared execution protocol. Each case fixes the visible verifier state, the admissible actions, the parser that maps model text into operators, and the oracle that scores progress along the trajectory (Table 1). Rows come from mainstream OR source tasks; difficulty comes from controlled modeling perturbations with explicit invariants; labels are replayed from typed verifier evidence. Within each controlled comparison, source rows and scoring rules are held fixed; changes to the model-facing output interface are declared explicitly. This fixed task and scoring protocol compares interpretation and repair-control behavior. The row, oracle, parser, and replay gates are materialized before model evaluation. The controller retrieves repair families keyed by RSS signatures. The retrieval key excludes source identifiers, oracle labels, and ground-truth repairs; the isolated-state controller study disables cross-run experience writes.

3.1 TYPED CASES WITH STRUCTURED VERIFIER STATE

Each OR-INSPECT-BENCH case is a record with five blocks. The *source block* captures OR domain, task text, data table, formulation, and provenance. The *failure block* stores difficulty, perturbation family, violated invariant, and repair target. The *verifier block* records solver/checker status, conflict signatures, rationality diagnostics, and parser outcomes. The *trace block* preserves visible history, candidate actions, verifier responses, and token/call accounting. The *oracle block* fixes the preferred diagnostic, next verifier action, and repair label. Every row therefore exposes the complete repair episode: *what the agent saw, what it did, how the verifier responded, and whether the action moved the repair in the intended direction*.

3.2 TWO EVALUATION LANES

TRACEJUDGE *fixes the trace.* The model receives a frozen, structured trace and answers six verifier-style questions: (i) preferred trace in a paired comparison; (ii) violated invariant; (iii) decisive step; (iv) correct next verifier action; (v) whether the endpoint rationale is rejected; and (vi) the counterfactual verifier prediction. This lane focuses on diagnostic ability without a live action budget. The main metric is the composite score (mean of six sub-judgements); strict-label agreement records exact matches with the deterministic oracle.

REPAIRRUN *executes repairs live.* The broad pipeline study uses four prompting/control configurations: `plain`, `cot`, `operator`, and the full verifier-native pipeline (CVL+RSS). Actions are constrained to a parsed operator schema so that every proposal is either executed or rejected by the verifier; free-text-only outputs receive parser status before scoring. The four modes share source

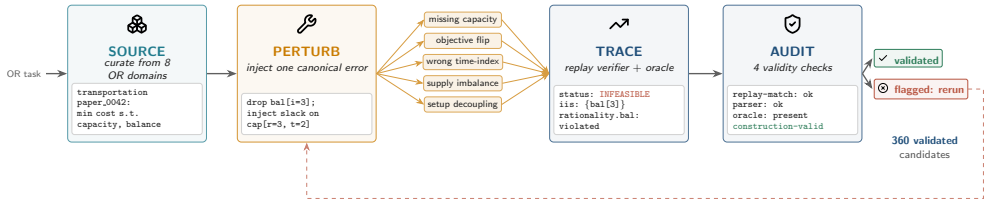


Figure 3: Four-gate OR-INSPECT-BENCH construction pipeline. A source OR task receives a controlled perturbation, verifier replay produces the trace and oracle action, and the final gate decides whether the row is construction-valid before model outputs are scored.

tasks, parser schema, and validity rules; their different interaction budgets are reported diagnostics, so these are system comparisons, not isolated component effects. The matched study adds raw-feedback multi-turn, RSS-only, and probe-free CVL conditions.

3.3 PAIRED STRICT AND SEMANTIC METRICS

STRICT_RRR is the primary live-repair metric. It requires a canonical repair target, correct `after` value, typed constraint reference, and accepted endpoint and rationality checks. Syntactically valid JSON alone is insufficient. The auxiliary SEMANTIC_RRR rule accepts concept-matched repairs that do not satisfy all canonical-output requirements. We label this original rule *semantic-v1*; it is a permissive diagnostic, not ground truth for semantic correctness. A recalibrated *semantic-v2* additionally checks objective-direction restoration (§5.3). Neither rule changes strict scores. A strict–semantic gap can reflect output constraints *or false semantic acceptance*, so it cannot alone separate formatting from reasoning. Malformed model outputs remain strict failures; exhausted provider retries are separately recorded reliability events.

3.4 BENCHMARK STATISTICS

The main slice contains **360 hardened cases** per (model, configuration) combination, drawn from seven OR mechanism families (Fig. 2: capacity allocation, flow balance, time index, integrality, probability normalisation, setup linking, skill assignment), with five canonical error types injected under controlled perturbation parameters and three hardening variants per base task. The 360 rows derive from 120 source tasks and are not treated as independent observations. The CVL+RSS live-repair lane evaluates this slice across **23** model snapshots for a total of $\sim$**33,000** trajectories. Source-task-clustered paired intervals quantify controller margins; eight process-family clusters provide a coarse sensitivity check (Appendix G). The external transfer suite contains five public OR benchmark slices (NL4Opt-245, MAMO-Easy subset with 100 rows, MAMO-Complex filtered subset with 203 rows, IndustryOR-100, and the LEAN-LLM-OPT 101-task subset) under each benchmark's native scoring protocol. Appendix Table 11 traces one admitted row from source task to oracle label, mirroring the raw-to-standardized examples used in the reference corpus.

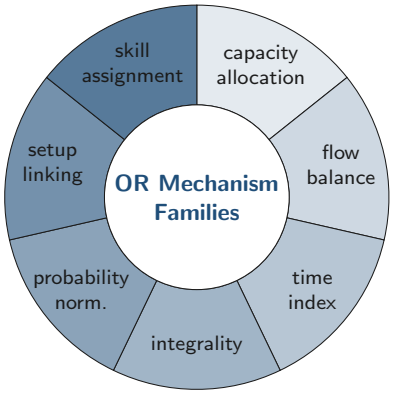


Figure 2: Seven OR mechanism families covered by OR-INSPECT-BENCH. The slice spans common trajectory-repair structures across domains.

4 BENCHMARK CONSTRUCTION

OR-INSPECT-BENCH construction follows a four-gate pipeline (Fig. 3) that turns OR source tasks into typed repair cases. The gates certify the source task, perturbation, oracle, verifier replay, and

provenance before any model output is scored. The released 360-instance task set is available in an anonymous repository.[1]

The construction target is selective coverage. Each admitted row must be both challenging and reproducibly labeled. Difficulty means a plausible local repair can move one verifier signal while damaging another OR invariant. Reproducibility means the label is replayed from typed solver/checker evidence; it does not imply independent certification of the oracle.

Gate 1 certifies source tasks. We draw tasks from mainstream OR domains: inventory and supply planning, transportation and network flow, facility location, assignment and workforce allocation, revenue management, scheduling and admission control, production lot sizing, and stochastic/SAA cases. A task must expose a recognizable OR invariant and a structured checker signal. Easy cases have a localized signature; medium cases require choosing among operators while preserving satisfied constraints; hard cases combine verifier channels or delayed rationality failures (Appendix B).

Gate 2 injects controlled errors. We inject perturbations from a fixed catalog of five canonical error types: missing capacity constraint, objective-direction flip, wrong time index, demand/supply imbalance, and setup–production decoupling. Every perturbation pairs with an expected verifier signature and a repair target. Local edits can restore one signal while damaging another OR invariant.

Gate 3 materializes traces and oracle labels. For fixed-trace diagnosis (TRACEJUDGE), the constructor materialises trace-derived questions and oracle-derived answer keys from stored verifier states — never from a free-form judge preference. For live repair (REPAIRRUN), the constructor materialises the same source task under four prompting modes (plain, CoT, operator, RSS+CVL) with explicit parser and budget settings. Mode-level differences then reflect the visible context, action interface, and controller.

Gate 4 finalizes row validity. The final gate separates construction validity from model-output validity. A row is excluded from capability aggregates for source malformation, missing oracle, failed verifier replay, or provider quota/transport errors outside the benchmark protocol. By contrast, malformed JSON, undeclared operators, wrong target syntax, and output-schema violations are model-output failures: they remain in the assigned-case denominator and count as strict repair failures, with parser-failure status available for diagnostics. Superseding reruns repair provider/runtime failures under a validity-priority rule fixed before scoring. The row record (§2.5) keeps the source, model, configuration, native outcome metrics, efficiency metrics, reliability status, and row identifier for every assigned case; the scoring fields are summarized in Appendix D.

A frozen 48-case composition set probes generalization beyond the main perturbation combinations. It holds out combinations while retaining the same primitive errors and source-task pool. Oracle replay checks construction consistency, while blinded model audits provide two complementary semantic checks: one calibrates acceptance rules and the other evaluates sampled repairs (Appendix G.6, Appendix G.7). Together, replay and model judgments provide distinct validation routes; domain-expert review can add domain-ownership evidence.

5 EXPERIMENTS

We separate system-level repair gains from the value of typed state and additional control. STRICT_RRR is the fraction of provider-valid assigned cases with a canonical, verifier-accepted repair; malformed outputs count as failures. Diagnostic SEMANTIC_RRR scores retain their rule version. TRACEJUDGE averages exact diagnostic-label agreement; external modeling accuracy (MA) uses each public benchmark's native rule. Appendix A.8 specifies denominators and gates.

5.1 FULL-PIPELINE REPAIR ACROSS 23 MODEL SNAPSHOTS

On the 360-row hardened slice, the full CVL+RSS pipeline improves over one-shot prompting for all 19 directly measured models in the displayed matrix (Fig. 4); four additional reference rows are imputed and marked separately. Large gains include `kimi-K2.5` (+63.6 pp), `Qwen3-30B-A3B` (+43.0), `Llama-3.3-70B-Instruct` (+41.4), and `DeepSeek-V3.2` (+39.2). The complete

[1] https://github.com/VerifierOR/or-inspect-bench

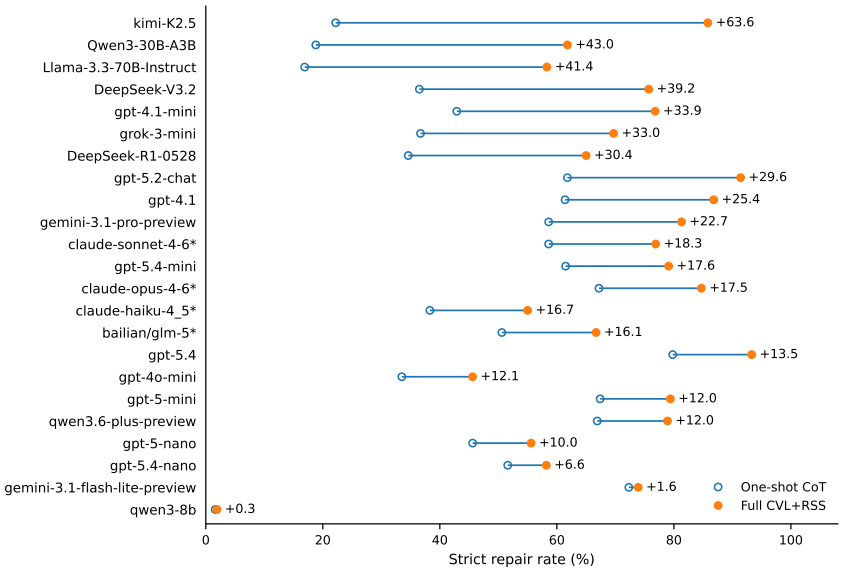


Figure 4: System-level repair across 23 model snapshots. Open markers show the CoT strict score and filled markers the full CVL+RSS pipeline. An asterisk marks an imputed reference row. Matched component comparisons in the text and Table 2 resolve typed feedback, output handling, and additional control effects.

Table 2: Matched RSS versus probe-free CVL on 360 cases per model. All values are strict repair percentages; Δ is in percentage points. Intervals use paired source-task clustering; each row compares fixed configurations.

Model	RSS	CVL	Δ	95% CI
Qwen3-30B-A3B	54.4	65.3	+10.8	[+5.6,+16.7]
GPT-5.4-nano	32.2	49.4	+17.2	[+12.5,+21.7]
GPT-5.4-mini	76.7	80.0	+3.3	[+0.6,+6.4]
GPT-4.1-mini	88.3	91.4	+3.1	[0.0,+6.7]
Kimi-K2.5	96.1	96.9	+0.8	[−1.4,+3.1]

system matrix and its per-model denominators are in Appendix D; subsequent matched studies resolve the effects of typed feedback, output normalization, and controller configuration.

5.2 Typed state and conditional control

The staged configuration screen holds source rows and scoring rules fixed. Compiling raw feedback into RSS adds 6.7/9.2/28.6 pp of strict recovery for GPT-5.4-mini/DeepSeek-V3.2/Qwen3-30B, respectively. The seven-mode screen separates this state-exposure gain from the model-dependent effects of additional control; the factorial below identifies canonical-emission correction as a reproducible Qwen component.

We next evaluate probe-free CVL against RSS on six model anchors in a run-isolated study. Each pair uses the same 360 problem IDs within its model's evaluation window, with experience writes disabled. We use 2,000 paired whole-cluster bootstrap draws (seed 42), grouping the three variants of each of 120 source tasks. Table 2 reports source-task intervals; eight-family sensitivity intervals are in Appendix G. Qwen and GPT-5.4-nano improve under both cluster definitions. The other anchor margins are smaller or unresolved, including a small negative DeepSeek point estimate. Together, the six anchors characterize model-conditional control margins and motivate validation on held-out cases.

The July Qwen pair uses 361 recorded LLM calls per condition; CVL uses 1,077,595 tokens versus 898,153 for RSS. Under the same candidate-generation condition, lexicographic selection reaches 65.3% strict recovery versus 28.6% for random selection. A separate, fresh three-arm Qwen run now directly compares RSS, RSS plus the same reference-free normalizer, and CVL-r2 on 360 identical cases. Strict all-error recovery is 110/360 (30.6%), 154/360 (42.8%), and 165/360 (45.8%), respectively. Adding the normalizer to RSS improves strict-RRR by 12.2 pp (source-task 95% CI [7.5, 16.9]; family CI [1.9, 23.1]); CVL-r2 adds a further 3.1 pp, with both intervals including zero. The matched three-arm study also records the interaction depth directly: all 360 RSS and RSS+normalizer cases use one LLM decision, while CVL-r2 uses more than one in 9/360 cases (8 use two calls; 1 uses three). Solver paths range from 4–6 calls in the two RSS arms and 3–7 in CVL-r2. Thus, this slice links the observed recovery differences to explicit output handling and verifier-path selection, with the additional full-package margin measured separately. The probe-by-canonicalization factorial below supplies a second component analysis within CVL configurations. Full paired intervals and per-arm call/token counts appear in Appendix G.

In a same-window Qwen 2×2 factorial, canonical-emission correction raises strict recovery by 14.4 pp with probes enabled and 20.6 pp with probes disabled; both effects have positive source-task- and family-clustered 95% intervals (Appendix Table 28). Probe-budget contrasts are unresolved at either output setting. This identifies canonical output handling as a load-bearing component within the tested CVL configurations.

5.3 Strict scoring and acceptance validation

Strict output requirements and OR correctness are evaluated separately. In an error-enriched 125-card audit, a blinded `claude-opus-4-5` judge produced 120 definite judgments. Agreement was 112/120 with strict scoring but only 16/120 with semantic-v1; many disagreements were explicit wrong objective directions. Requiring direction restoration raised semantic-v2 agreement to 108/120 on the same audit set, with strict scores unchanged. The appendix reports the sampling design and labels legacy semantic-v1 scores explicitly.

A second audit evaluates Qwen's RSS+normalizer repairs on a fresh random sample. From the 243 cases not represented in the prior adjudication, we sampled 80 without replacement and asked GPT-6 Astra to judge each repair without seeing its benchmark score, injected-error label, reference target, or prior audit label. The model accepted 12 repairs, rejected 37, and marked 31 uncertain. Among the 49 decisive judgments, 42 agreed with strict RRR (85.7%). All 12 accepted repairs passed strict scoring; 30 of the 37 rejected repairs failed strict scoring, while seven strict passes received a semantic rejection. This sampled audit provides a blinded, model-based semantic-validation route alongside strict scoring and verifier replay. We report its outputs as model judgments.

Holding typed RSS input and budget settings fixed, a separate Qwen output-interface diagnostic changes strict recovery from 50.3% to 99.4% when free-form output is mapped to benchmark reference actions. This answer-assisted projector quantifies schema sensitivity; the reference-free normalizer evaluated above instead operates directly on generated outputs.

5.4 TraceJudge verifier-judgement agreement

Table 3 reports the top-12 diagnostic rows, each evaluated on 96 de-duplicated paired traces. High-scoring systems include `gpt-5.4` (80.0), `gpt-5.2-chat` (79.9), `claude-opus-4-6` (75.0), and `gemini-3.1-pro-preview` (71.7). This lane measures recognition of violated invariants, decisive steps, and next verifier actions without granting a live repair budget. Structural and hard-action subscores appear in the appendix.

Table 3: TraceJudge fixed-trace judgement. Top-12 systems by row-level TraceJudge score. The score (%) is the six-part exact-match composite after de-duplicating each model/problem pair and applying the fixed validity-priority replacement rule when a source call failed. The full system table with structural breadth and compact token counts is in Appendix D.

System	Diagnostic score
gpt-5.4	80.0
gpt-5.2-chat	79.9
claude-opus-4-6	75.0
gemini-3.1-pro-preview	71.7
gpt-5-nano	70.3
gpt-4.1-mini	69.3
kimi-K2.5	68.6
gpt-5.4-nano	66.3
gpt-5.4-mini	65.8
grok-3-mini	64.4
DeepSeek-V3.2	62.0
gpt-4.1	59.9

5.5 Composition holdout and external benchmarks

On 48 frozen new perturbation compositions, CoT/RSS/CVL strict scores are 10.4/41.7/64.6% for Qwen, 37.5/62.5/75.0% for GPT-5.4-mini, 29.2/54.2/54.2% for DeepSeek, and 47.9/100/100% for Kimi. The different margins again favor conditional rather than uniform adoption. The holdout reuses source tasks and primitive errors while holding out new perturbation compositions, providing recombination evidence within the benchmark. A generic free-form agent adds an interface-portability comparison alongside the matched controller studies (Appendix G).

The five public benchmark results in Table 4 use their native formulation-acceptance rules and are reported alongside published reference ranges. Matched controller comparisons use the within-study protocols above; Appendix E records the external benchmark details.

Table 4: Contextual public-benchmark evaluation. Our runs use the listed benchmark subsets and native MA rules. Reference ranges are source-reported, not rerun under a common harness or output policy (Huang et al., 2025a; Xiao et al., 2024; AhmadiTeshnizi et al., 2024; Liang et al., 2026).

Benchmark	N	Pipeline MA (%)	Literature range
LEAN-LLM-OPT-101	101	88.1	5.0–85.1
NL4Opt	245	98.4	47.3–97.1
MAMO-Easy	100	95.0	66.5–93.5
MAMO-Complex	203	85.7	14.6–74.9
IndustryOR	100	80.0	27.0–65.0

5.6 Scope of the evidence

The evidence connects system-level repair gains with matched component comparisons: typed-state exposure improves on raw feedback, canonical-emission correction improves strict recovery under both Qwen probe settings, and the new three-arm comparison measures the normalization contribution directly. Across six paired model anchors, additional control margins vary in magnitude; source-task and process-family intervals report their paired uncertainty.

6 Related Work

OR-LLM benchmarks establish solver-graded formulation as an evaluation target, including NL4Opt, MAMO, LEAN-LLM-OPT, ORLM, OptMATH, and DP-Bench (Ramamonjison et al., 2023; Huang et al., 2025b; Liang et al., 2026; Huang et al., 2025a; Lu et al., 2025; Zhou et al., 2025). Recent OR repair work addresses complementary questions. OptiRepair studies IIS-guided feasibility repair together with five inventory-theoretic rationality checks over 976 supply-chain problems and 22 API models (Ao et al., 2026a). ORLoopBench formulates solver-in-the-loop repair as an MDP and evaluates 5,362 LP/MILP repair instances alongside behavioral rationality benchmarks (Ao et al., 2026b). Our study instead characterizes the inference-time repair interface: a typed five-channel verifier state, a controlled comparison of feedback representation, output normalization, and

controller configurations, and an evaluation design that pairs fixed-trace diagnosis with live repair. This combination lets us measure which interface components change strict recovery on matched cases and how the effects vary across models and repair regimes.

General agents use tool feedback for reflection and search (Yao et al., 2023b; Shinn et al., 2023; Madaan et al., 2023; Gou et al., 2024; Yao et al., 2023a; Zhou et al., 2024; Jimenez et al., 2024; Liu et al., 2024; Qin et al., 2024), while RLVR and process-reward methods train policies or step judges (DeepSeek-AI, 2025; Lambert et al., 2024; Shao et al., 2024; Yu et al., 2025; Wang et al., 2024; Lightman et al., 2024; Luo et al., 2024). We connect typed OR state, verifier transitions, and trajectory-level evaluation in an inference-time repair interface. Appendix F expands the comparison.

7 Discussion and Conclusion

RSS, CVL, and OR-Inspect-Bench make verifier state, repair control, and trajectory outcomes jointly measurable. Matched controls show gains from typed feedback over raw multi-turn input and a 12.2 pp strict-RRR gain from adding reference-free output normalization to RSS on Qwen. The full CVL-r2 configuration adds a further 3.1 pp on the same cases, with paired intervals that include zero; across six model anchors, the additional control margin varies by model. Together, these results identify output normalization as a reliable component in the tested Qwen setting and quantify the separate, model-conditional contribution of additional control.

OR-Inspect-Bench pairs live repair with fixed-trace diagnosis using solver/domain replay, controlled perturbations, and an acceptance-rule audit informed by a blinded LLM adjudicator. A fresh random sample of 80 Qwen RSS+normalizer repairs adds a second semantic check: GPT-6 Astra agrees with strict scoring on 42 of 49 decisive judgments. Together with deterministic replay and the acceptance-rule audit, this connects construction consistency, score calibration, and repair semantics. The composition holdout evaluates new combinations of the benchmark's source tasks and primitive errors. Domain-expert review, new OR domains, and deployment traces can extend this model-based validation route to domain-owned cases and operational use.

AI USE STATEMENT

Generative AI tools assisted with research planning, manuscript organization, language editing, LaTeX preparation, experimental-code development, and interpretation of stored results. These research-assistance uses are distinct from the LLM repair agents being evaluated and the blinded model adjudicators described in Appendix G.6 and Appendix G.7. Numerical claims in this paper are tied to recorded evaluation results. The authors are responsible for the final text, experimental claims, and artifacts, including AI-assisted content.

REPRODUCIBILITY STATEMENT

Appendix A specifies the state representation and controller; Appendix C records the schema and parser contract; Appendix B documents benchmark construction; and Appendix G separates evaluation campaigns, configurations, paired inference, output-interface diagnostics, and acceptance auditing. The public task release contains the benchmark instances. Model snapshots, prompts, budgets, and recorded outputs are required to reproduce the additional controlled comparisons; the public task release alone is not the full execution artifact.

REFERENCES

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*, volume 235, pp. 577–596, 2024.

Ruicheng Ao, David Simchi-Levi, and Xinshang Wang. OptiRepair: Closed-loop diagnosis and repair of supply chain optimization models with LLM agents. *arXiv preprint arXiv:2602.19439*, 2026a.

Ruicheng Ao, David Simchi-Levi, and Xinshang Wang. ORLoopBench: Solver-in-the-loop benchmarks for self-correction and behavioral rationality in operations research. *arXiv preprint arXiv:2601.21008*, 2026b. Version 3; accepted by ICML 2026.

Dimitris Bertsimas and Georgios Margaritis. Robust and adaptive optimization under a large language model lens. *arXiv preprint arXiv:2501.00568*, 2024.

Gary E. Bolton and Elena Katok. Learning-by-doing in the newsvendor problem: A laboratory investigation of the role of experience and feedback. *Manufacturing & Service Operations Management*, 10(3):519–538, 2008. doi: 10.1287/msom.1060.0190.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *ICLR*, 2024.

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. *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2678–2710, 2025b. doi: 10.18653/v1/2025.findings-naacl.146.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Mirko Kremer, Brent Moritz, and Enno Siemsen. Demand forecasting behavior: System neglect and change detection. *Management Science*, 57(10):1827–1843, 2011. doi: 10.1287/mnsc.1110.1382.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M. Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, 2025.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Kuo Liang, Yuhang Lu, Jianming Mao, Shuyi Sun, Chunwei Yang, Congcong Zeng, Xiao Jin, Hanzhang Qin, Ruihao Zhu, and Chung-Piaw Teo. Large-scale optimization model auto-formulation: Harnessing LLM flexibility via structured workflow. *arXiv preprint arXiv:2601.09635*, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *ICLR*, 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating llms as agents. In *ICLR*, 2024.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. *arXiv preprint arXiv:2502.11102*, 2025.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models with automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 2023.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *ICLR*, 2024.

Rindranirina Ramamonjison, Timothy T. Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 Competition Track*, 2023.

Maurice E. Schweitzer and Gérard P. Cachon. Decision bias in the newsvendor problem with a known demand distribution: Experimental evidence. *Management Science*, 46(3):404–420, 2000.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Raghav Thind, Youran Sun, Ling Liang, and Haizhao Yang. OptimAI: Optimization from natural language using LLM-powered AI agents. *arXiv preprint arXiv:2504.16918*, 2025.

Guiyao Tie, Zenghui Yuan, Zeli Zhao, Chaoran Hu, Tianhe Gu, Ruihang Zhang, Sizhe Zhang, Junran Wu, Xiaoyue Tu, Ming Jin, Qingsong Wen, Lixing Chen, Pan Zhou, and Lichao Sun. Can LLMs correct themselves? a benchmark of self-correction in LLMs. In *Proceedings of the NeurIPS 2025 Datasets and Benchmarks Track*, 2025.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-Shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 9426–9439, 2024. doi: 10.18653/v1/2024.acl-long.510.

Shirley Wu, Michel Galley, Baolin Peng, Hao Cheng, Gavin Li, Yao Dou, Weixin Cai, James Zou, Jure Leskovec, and Jianfeng Gao. CollabLLM: From passive responders to active collaborators. In *International Conference on Machine Learning*, 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. *International Conference on Learning Representations*, 2024.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023b.

Gregory Yauney, Shahzaib Saqib Warraich, and Swabha Swayamdipta. How reliable is language model micro-benchmarking? In *International Conference on Learning Representations*, 2026.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Weinan Dai, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *NeurIPS*, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, 2022.

Jiahan Zhang, Muqing Jiang, Nanru Dai, Taiming Lu, Arda Uzunoglu, Shunchi Zhang, Yana Wei, Jiahao Wang, Vishal M. Patel, Paul Pu Liang, Daniel Khashabi, Cheng Peng, Rama Chellappa, Tianmin Shu, Alan Yuille, Yilun Du, and Jieneng Chen. World-In-World: World models in a closed-loop world. In *ICLR*, 2026a.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.

Xinchen Zhang, Xiaoying Zhang, Youbin Wu, Yanbin Cao, Renrui Zhang, Ruihang Chu, Ling Yang, Yujiu Yang, and Guang Shi. Generative universal verifier as multimodal meta-reasoner. In *ICLR*, 2026b.

Yunxiang Zhang, Muhammad Khalifa, Shitanshu Bhushan, Grant D. Murphy, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. MLRC-Bench: Can language agents solve machine learning research challenges? In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

Xuhua Zhao, Yuxuan Xie, Caihua Chen, and Yuxiang Sun. AIM-Bench: Evaluating decision-making biases of agentic LLM as inventory manager. *arXiv preprint arXiv:2508.11416*, 2025.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *ICML*, 2024.

Chenyu Zhou, Jingyuan Yang, Linwei Xin, Yitian Chen, Ziyan He, and Dongdong Ge. Auto-formulating dynamic programming problems with large language models. *arXiv preprint arXiv:2507.11737*, 2025.

APPENDIX OVERVIEW

The appendix proceeds in the same order as the paper's evidence pipeline. Appendix A specifies the RSS state, CVL controller, metrics, and runtime settings. Appendix B gives the OR-INSPECT-BENCH construction pipeline, source-domain taxonomy, perturbation examples, and construction/scoring separation. Appendix C records prompt templates and parser rules. Appendix D reports the full OR-INSPECT-BENCH matrices, strict/semantic diagnostics, difficulty anchor, and ablations. Appendix E gives the public-benchmark transfer tables, and Appendix F expands the related-work comparison. Appendix G collects the matched controller study, component diagnostics, and scoring audit.

A METHOD AND IMPLEMENTATION DETAILS

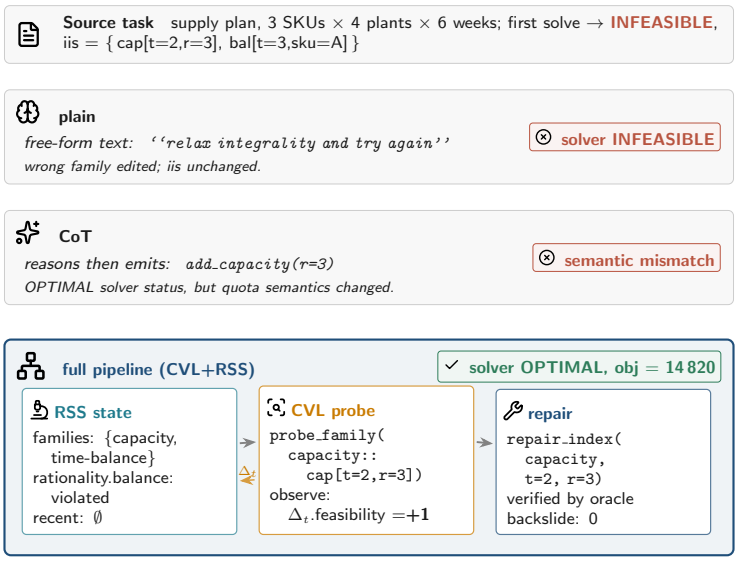


Figure 5: Compact view of the verifier-native repair loop. The diagram emphasizes the same state–action–verifier transition as Figure 1, with the visual focus on the typed handoff between RSS, candidate actions, verifier feedback, and the next repair state.

A.1 REPAIR-SUFFICIENT STATE

The repair-sufficient state z_t contains solver status, conflict or slack signatures, constraint-family labels, rationality flags, parser status, recent interventions, candidate operators, and budget counters. The state is compact: it preserves the verifier signals that distinguish productive repairs from irrelevant edits.

Table 5: RSS fields used by the reported experiments.

Field group	Examples	Downstream use
Solver state	feasible, infeasible, unbounded, objective value	Endpoint and recovery status
Conflict state	IIS family, slack pattern, violated balance	Operator retrieval and probe choice
Rationality state	semantics flags, service checks, probability checks	Reject false feasibility
History state	prior probes, repairs, backslides, budget	Efficiency and search control
Parser state	schema valid, unknown operator, output cap	Validity and runtime status

A.2 Operator Vocabulary

The shared operator vocabulary is grouped by role:

Inspection `inspect_conflict`, `probe_family` expose the active verifier family.

Repair `repair_index`, `restore_integrality`, `repair_objective`, `add_capacity_family`, `normalize_scenarios` edit typed formulation fields.

Terminal `submit_repair` is the only terminal operator.

Domain adapters may expose narrower parameter sets, but every run stores the vocabulary used by the prompt and parser.

A.3 Controller configurations

The full CVL configuration exposes diagnostic probes and repairs to a verifier-backed search. In the matched comparisons, CVL uses the same search infrastructure without probes and adds an explicit normalization step between the model response and the benchmark parser. The two configurations are not interchangeable. The full pipeline supplies a 23-row system matrix with 19 directly measured models and four marked imputed reference rows; the isolated-state six-anchor comparison tests CVL against RSS-only interaction.

The normalizer applies one-to-one aliases for schema keys, canonicalizes case/punctuation-equivalent public fields, and constructs `constraint_ref=field::target` only when the emitted field and target already agree with the public schema. It may fill an omitted or blank `after`: objective direction is derived from the visible corrupted artifact, and other supported fields use public schema literals. It does not replace any explicit value, including an explicitly wrong objective direction. It has no access to problem IDs, reference repairs, labels, or scorer verdicts. Because it can complete missing repair parameters, it is a substantive output-normalization aid rather than a cosmetic parser change.

The ordinary RSS-only baseline does not use this normalizer or search. Consequently, its comparison with CVL identifies the combined additional configuration. Ranking is separately compared to random selection, and the reference-action projector in Appendix G.5 is a different, benchmark-specific diagnostic. No exhaustive factorial attribution is claimed.

A.4 Executed state projection and transition reward

For an adapter-backed node n, let

$$\phi_{\mathrm{impl}}(n) = (p(n), e(n), g(n)).$$

Here p is the adapter quality-check pass indicator; e is an exposed exact-match metric, or p when absent; g is the negative objective gap, or 0 for a passing node and $-\infty$ otherwise when no gap is exposed. The OR-Inspect adapter does not expose exact-match or objective-gap coordinates, so this preference projection is $(p, p, 0)$ on passing nodes and $(0, 0, -\infty)$ otherwise. Adapter quality pass denotes checker recovery without false acceptance; it is not the benchmark's strict canonical-repair score. The latter is evaluated separately. In particular, we do not subtract these extended-real tuples to calculate a reward.

The finite transition features use parent snapshot b and child snapshot a:

$$\Delta_{\mathrm{feas}} = \frac{\mathrm{IISgap}(b) - \mathrm{IISgap}(a)}{\max(1, \mathrm{IISgap}(b))}, \tag{3}$$

$$\Delta_{\mathrm{rat}} = \mathrm{rat}(a) - \mathrm{rat}(b). \tag{4}$$

IISgap is the adapter-provided IIS-gap diagnostic and rat its rationality diagnostic; neither is a language-model self-rating. The OR-Inspect transition scorer uses parsimony -0.05, cost 0, and objective quality $\mathbf{1}[\mathrm{OPTIMAL}]$. Its backed-up reward is

$$s(a; b) = \Delta_{\mathrm{feas}} + \Delta_{\mathrm{rat}} - 0.05 + \mathbf{1}[\mathrm{OPTIMAL}] + 1, \tag{5}$$

$$r(a; b) = \mathrm{clip}_{[0,1]}\left(s(a; b)/2\right). \tag{6}$$

This scalar reward, the traversal policy, and the final preference key have different roles.

Algorithm 1 Verifier-guided search for CVL

Require: adapter state, model, configured operators, solver-call budget B
Compile the root snapshot and RSS; initialize the search tree
while solver calls $< B$ and configured safety limits are not reached **do**
  Select a leaf using UCT (Thompson only if enabled)
  if the leaf is terminal **then**
    Back up its adapter-quality pass indicator
  else
    Generate unique candidates in serialized order, up to capacity
    In the probe-free configuration, disable probes and normalize the repair response
    for candidate a while the solver-call budget remains **do**
      Clone the adapter; apply a; capture the child snapshot
      Back up the scalar transition reward $r(\text{child}; \text{parent})$
    end for
    if a child passes adapter quality **then**
      break from the search loop
    end if
  end if
end while
Select a successful leaf by mean reward, else follow K greedily
return root-to-leaf action path and verifier records

A.5 Search and extraction

UCT selects nodes during traversal; optional Thompson mode is used only when its flag is enabled. The greedy child-preference key is

$$K(n) = \operatorname{lex}\big(p(n), e(n), g(n), \mathbf{1}[\text{terminal success}], \mathbf{1}[\neg\text{terminal bad}], \bar{r}(n), v(n)\big),$$

where $\bar{r}$ is mean backed-up reward and v is visit count. Exact key ties retain serialized child order. If a visited terminal-success node exists, extraction chooses one with maximal mean reward (first encountered on a tie in the extraction traversal). Otherwise it follows the greedy key. The construction oracle's ranking is not this runtime controller key.

A.6 Closed-Loop Example

A live-repair trace begins with an infeasible transportation model whose verifier flags a violated flow-balance family. The RSS compiler records the infeasible status, the balance-family signature, and that capacity checks are satisfied. The controller rejects a capacity-relaxation repair because it risks changing a satisfied family, chooses a balance probe, observes that the verifier delta localizes the error to a time-indexed conservation constraint, and then submits an index repair. The final metric row records endpoint recovery, rationality recovery, two verifier calls, one accepted operator, no backslide, and no reliability failure.

A.7 Memory and version scope

The isolated-state evaluations use a fresh run-scoped experience store, with online writes disabled and no bootstrap trajectory snapshot. Operator retrieval may be invoked on eligible cases, but previous evaluation trajectories are not accumulated across runs. This protocol was introduced after identifying cross-run sensitivity of a mutable experience store; affected exploratory runs are not used for the matched controller claims. The broad pipeline study and configuration screen remain separately identified in Appendix G. Model/provider, parser and normalizer versions, probe and knowledge-base flags, branching/depth caps, candidate order, and benchmark manifest are needed to identify an execution.

A.8 METRIC DEFINITIONS

Let i index a valid row and let $\mathbf{1}[\cdot]$ be the indicator. The main process metrics are

$$\text{RR} = \frac{1}{N}\sum_i \mathbf{1}[\text{endpoint accepted}_i], \tag{7}$$

$$\text{RRR} = \frac{1}{N}\sum_i \mathbf{1}[\text{endpoint accepted}_i \wedge \text{rationality accepted}_i], \tag{8}$$

$$\text{OVG} = \frac{1}{N}\sum_i \mathbf{1}[\text{oracle action selected}_i], \tag{9}$$

$$\text{FalseAccept} = \frac{1}{N}\sum_i \mathbf{1}[\text{endpoint accepted}_i \wedge \neg\text{oracle accepted}_i]. \tag{10}$$

Agreement metrics compare protocol labels to deterministic oracle labels. Compact diagnostics report checker calls and token volume when they help interpret mechanism behavior. Wall-clock duration, provider cost, and model-call counts remain in row-level records; cost comparisons must use the configuration-specific accounting rather than assume equal inference work.

STRICT_RRR instantiates RRR on provider-valid live-repair rows. A row with a recorded provider response and positive token count is provider-valid; zero-token provider failures are reported separately and excluded from capability denominators. Malformed model output on a provider-valid row remains in the denominator and receives zero strict and semantic credit. In pooled system-matrix summaries, repeated run records are aggregated as rows rather than de-duplicated by problem ID. A row receives strict credit only when the final parsed action satisfies the declared parser schema, selects the accepted repair target, and passes both endpoint and rationality checks under replay. SEMANTIC_RRR uses the same denominator and endpoint/rationality gates, and credits additional concept-matched repair outputs. This permissive semantic-v1 rule is not an independent semantic oracle. Semantic-v2 further checks objective-direction restoration; its calibration is reported in Appendix G.6. TRACEJUDGE is computed on fixed-trace diagnostic rows after row-level de-duplication; each sub-score is an exact match between a model-provided diagnostic field and the deterministic oracle label for that field. Public MA is computed only on the external benchmark denominator named in the corresponding public source; a task receives credit when the final formulation is accepted by that benchmark's native parser/solver/scoring rule. Checker-call and token columns are per-case averages over the same rows used by the adjacent score column.

Table 6: Metric-to-record mapping. Field names follow the run schema used for the reported experiments.

Metric	Required row fields	Reported quantity
Endpoint success	final verifier status, validity flag	Fraction of rows whose final formulation is accepted
Rational recovery	endpoint status, rationality flags	Fraction of rows accepted by endpoint and rationality checks
Verifier-state progress	consecutive RSS signatures, verifier delta	Change in typed verifier state between consecutive accepted actions
Oracle agreement	model answer/action, oracle label	Exact-match agreement with deterministic oracle labels
Backslide rate	prior satisfied families, later verifier state	Fraction of rows in which a later action breaks a previously satisfied family
Compact diagnostics	checker calls, token volume	Per-case mean solver/checker/verifier calls and total LLM tokens
Reliability ledger	timeout, quota, transport class	Provider failures tracked separately; malformed model output counts as failure

A.9 Hyperparameters And Runtime Settings

Table 7 lists the full-pipeline settings. The isolated-state comparisons use their recorded configuration overrides, including disabling probes and experience writes as specified above. A nominal budget does not imply equal realized LLM calls, verifier calls, or tokens.

Table 7: CVL+RSS runtime settings. Timeouts are wall-clock per LLM subcall, budget caps are per source task, and retry caps gate provider reliability events before a row is marked invalid.

Setting	Value	Role
Search budget B	≤ 12 verifier rounds	Full-pipeline budget setting; realized calls recorded separately
Branch budget	up to 8 alternatives	Candidate actions scored per round
Code-generation timeout	120 s	Default per-call wall-clock for formulation branches
Structural follow-up timeout	20 s	Wall-clock for branches after a wrong-first OPTIMAL
Spec extraction timeout	20 s	Wall-clock for table/spec extraction subcalls
Sanity-check timeout	20 s	Wall-clock for GT-free verifier sanity probes
Repair max tokens	4096	Cap on completion tokens per repair turn
Consecutive timeout cutoff	2	Abort branch after this many timeout-like failures in a row
Provider retry cap	3	Maximum chat attempts before a row is marked reliability-invalid
Solver thread cap	1	Per-call solver thread limit (LEAN/Gurobi stability)

A.10 Runtime Accounting

Malformed schema outputs are parse failures and therefore strict capability failures for the assigned provider-valid case. Nonretryable provider errors, quota exhaustion, and transport failures are reliability events. Capability tables count malformed model outputs as repair failures and keep provider availability outside the repair score.

B Benchmark Construction Details

OR-Inspect-Bench materializes a typed benchmark object with 120 source tasks, 240 controlled candidates, 18,679 replay-derived candidate traces, and no source-validation or reference-solution failures in the reported denominator.

B.1 Source Task Schema

Each source task stores: domain, natural-language description, data table, canonical formulation or executable scaffold, declared checks, reference solution, admissible verifier channels, and provenance. A source task enters the benchmark only if all declared checks replay and a reference solution is available.

Table 8: OR-Inspect-Bench row construction pipeline. Each evaluated row is a sequence of typed objects: source task, perturbation, verifier replay, oracle label, and evaluation row.

Stage	Stored object	Role in evaluation
Source task	OR domain, data table, canonical formulation, checks, reference solution	Confirms that the row comes from a recognizable OR problem family
Controlled perturbation	Failure family, violated invariant, difficulty, target repair	Makes benchmark difficulty explicit and comparable
Verifier replay	Solver status, checker response, rationality diagnostic, visible state	Grounds labels in deterministic evidence
Oracle label	Preferred diagnostic, next verifier action, accepted repair operator	Defines the dispute-minimized target for trace and repair scoring
Evaluation row	Prompt mode, parser schema, validity flag, action trace, strict/semantic metrics	Uses the same row record for cross-model and cross-mode comparisons

B.2 Domain And Difficulty Design

The benchmark covers eight mainstream OR domains: assignment/workforce, facility location/capacity, inventory/supply planning, production lot sizing, revenue management/assortment, scheduling/admission control, stochastic SAA, and transportation/network flow. Each domain has 15 source tasks: five easy, five medium, and five hard. Repair structure determines difficulty. Easy cases have one localized verifier signal, medium cases require preserving another satisfied family while repairing the target, and hard cases combine delayed rationality or counterfactual probe effects.

Table 9: OR-Inspect-Bench source-domain taxonomy. Domains, typical invariants, and oracle evidence are listed in one place.

Domain	Typical invariant family	Verifier evidence used by the oracle
Assignment/workforce	one-to-one assignment, skill compatibility, availability	matching residuals, uncovered demand, incompatible pair flags
Facility/capacity	facility opening, capacity, service compatibility	capacity slack, open/serve links, compatibility flags
Inventory/supply planning	flow conservation, service level, capacity by period	balance residual, stockout/service flag, period-index conflict
Production lot sizing	setup-production linking, integrality, inventory balance	fractional setup, missing big-M link, carryover residual
Revenue/assortment	offer-set feasibility, probability normalization, objective direction	probability-mass error, dominated assortment, sign-flipped objective
Scheduling/admission	time-index precedence, resource capacity, admission logic	precedence violation, resource overload, rejected interval
Stochastic SAA	scenario weights, nonanticipativity, expected objective	weight sum, scenario consistency, expectation mismatch
Transportation/network flow	supply-demand balance, arc capacity, conservation	node imbalance, saturated arc slack, flow-conservation residual

The construction pass treats a source task as valid when its reference solution and declared checks replay without manual intervention. This mirrors the human-validation role in benchmark papers but uses deterministic OR records: source-level replay, perturbation-level oracle replay, and row-level parser/verifier replay. Tasks that fail any check remain outside capability denominators.

B.3 Process Families

The eight process families are endpoint-tie/process split, feasible-rationality split, backslide, late recovery, perturbation stability, probe-value split, verification-cost split, and boundary semantics. These families are chosen to expose precisely the cases where endpoint scoring can tie or misrank trajectories.

B.4 Construction Algorithm

For each source task, the constructor selects a controlled failure family, materializes candidate repairs or trace comparisons, executes deterministic validators, and writes a case card. A candidate enters reported aggregates only if the oracle label, visible verifier state, parser schema, source provenance, and replayed verifier decision are all present. The construction pass marks the benchmark construction-ready, with one warning: hardening traces with parse failures or output-cap hits must remain separate from model repair capability metrics.

B.5 Representative Case Pattern

A typical inventory case exposes a feasible endpoint that violates a service or balance invariant. Endpoint scoring accepts both candidates, but OR-Inspect-Bench distinguishes the candidate that restores the balance equation from one that relaxes inventory conservation. Similar patterns appear in facility capacity, assignment availability, revenue assortment, scheduling admission, production setup, stochastic normalization, and transportation flow-balance cases.

Table 10: OR-INSPECT-BENCH process families and representative OR interpretation.

Family	Representative interpretation
Endpoint tie/process split	Two feasible endpoints differ in whether the repair used decisive verifier evidence
Feasible-rationality split	Solver accepts a solution that violates service, balance, or probability semantics
Backslide	A later edit loses a previously repaired feasibility or rationality condition
Late recovery	Early actions appear unproductive but reveal the family needed for final repair
Perturbation stability	The preferred action remains correct under a targeted data perturbation
Probe-value split	A diagnostic probe is better than an immediate but low-value repair
Verification-cost split	Two valid trajectories differ by unnecessary solver/checker calls
Boundary semantics	A syntactically small edit crosses an OR modeling boundary, such as capacity versus compatibility

Table 11: Source-to-oracle case card for one admitted OR-INSPECT-BENCH row. The card shows the source object, controlled perturbation, verifier replay, candidate actions, oracle label, and validity rule.

Field	Example value
Source object	Facility-assignment instance with customer demand, facility capacities, and a compatibility matrix; reference solution opens facilities A/B and assigns every customer to a compatible open facility.
Controlled perturbation	Drop one compatibility row after an infeasible capacity repair, creating a feasible-looking assignment that serves customer i from incompatible facility A.
Verifier replay	MILP status is OPTIMAL; capacity slack is nonnegative; RSS records `rationality.compatibility = violated` and points to `compatibility::facility_A`.
Candidate action A	`relax_capacity(facility_A)`; endpoint remains feasible but the decisive compatibility violation is unchanged.
Candidate action B	`submit_repair(compatibility::facility_A)`; adds the missing compatibility constraint while preserving the already satisfied capacity family.
Oracle label	Prefer B. The accepted repair is the one whose verifier delta changes rationality from violated to ok without damaging feasibility or objective direction.
Validity rule	The row is construction-valid only if source replay, perturbation replay, the declared parser schema, and post-repair verifier replay all match this card; malformed model outputs are scored as strict failures.

The card lists the prompt object, the controlled failure, the evidence channel behind the label, and the reason endpoint scoring can be misleading. The model may see different subsets of the card depending on whether the lane is fixed-trace inspection or live repair, but the oracle and denominator gates are identical.

B.6 CONSTRUCTION AND SCORING SEPARATION

OR-INSPECT-BENCH is native to verifier-mediated repair. The benchmark rows are selected before CVL outcomes are evaluated. The admitted source tasks, perturbation templates, oracle labels, parser schemas, and replay gates are materialized before model evaluation. Operator names are domain-level repair families (for example, capacity, compatibility, balance, integrality), separated from problem identifiers or reference repairs. The operator memory used by CVL+RSS is keyed by

RSS signatures and stores repair families without row-specific answers; the key excludes source ids, oracle labels, and ground-truth repaired formulation. The matched lightweight-controller study uses a fresh run-local experience store and disables online writes. Its isolation protocol is documented in Appendix G; it must not be assumed retroactively for every broad-study execution. Baselines share source rows and validity rules within each declared comparison. Scoring-rule versions and the separate output-projector diagnostic are identified explicitly rather than treated as one common interface.

This separation keeps the benchmark and controller roles explicit. CVL is a reference controller for the verifier-state interface; OR-INSPECT-BENCH fixes the row, oracle, parser, and replay protocol for any evaluated controller.

B.7 CONSTRUCTION REFINEMENT

Construction follows a benchmark-curation loop. Candidate rows first pass source validation and reference-solution validation. They then pass oracle validation, parser-schema validation, and replay validation. Rows with ambiguous oracle labels, non-replayable verifier states, provider outages, or output-cap contamination remain outside reported aggregates until a clean attempt exists. Malformed model-generated JSON or undeclared actions are different: they are preserved as parser-failure evidence and count as strict repair failures for the provider-valid assigned case.

B.8 CONCRETE PERTURBATION EXAMPLES

Each perturbation family is materialised through a deterministic recipe over a parameterised source task. We give one concrete instance per family; the released catalog contains 24 such templates per family across the eight domains. Every box shows the literal verifier signature before and after the perturbation, plus the oracle-preferred repair operator keyed by the RSS signature.

⚠ Perturbation: missing capacity

domain: supply_planning | difficulty: easy
canonical form: $\sum_i x_{i,t} \le C_{r,t} \quad \forall r, t$
perturbation: drop the capacity-family constraint for $(r{=}3, t{=}2)$
verifier signature (post-perturb):
status = OPTIMAL
iis = $\emptyset$, but rationality.balance = violated,
rationality.feasibility (semantic) = False
oracle repair: restore_capacity(r=3, t=2)

⚖ Perturbation: supply-demand imbalance

domain: transportation | difficulty: medium
canonical form: $\sum_j y_{i,j} = s_i \quad \forall i$
perturbation: replace s_3 with $s_3 - \delta$, $\delta > 0$
verifier signature: status = INFEASIBLE, iis = {bal[i=3]}
oracle repair: repair_balance(target=i=3, restore=$+\delta$)

📅 Perturbation: wrong time-index

domain: scheduling | difficulty: medium
canonical form: $z_{i,t+1} \ge z_{i,t} \quad \forall i, t$
perturbation: shift the index by -1 in one constraint family
verifier signature: status = OPTIMAL but rationality.monotonicity = violated, $z_{i,t}$ decreases between $t{=}3$ and $t{=}4$
oracle repair: repair_index(family=monotonicity, $\Delta t = +1$)

Perturbation: integrality relaxed

domain: production_lot_sizing | difficulty: medium
canonical form: $z_{i,t} \in \{0,1\}$ (setup indicator)
perturbation: relax to $z_{i,t} \in [0,1]$
verifier signature: status = OPTIMAL with rationality.integrality = relaxed; objective dominated by fractional setups
oracle repair: restore_integrality(z[i,t])

Perturbation: setup–production decoupled

domain: production_lot_sizing | difficulty: hard
canonical form: $x_{i,t} \leq M \cdot y_i \quad \forall i,t$ (linking)
perturbation: drop the linking constraint between setup y_i and production $x_{i,t}$
verifier signature: status = OPTIMAL with rationality.linking = violated; $x_{i,t} > 0$ while $y_i = 0$
oracle repair: repair_link(setup=y[i], production=x[i,t])

C Prompt Templates and Parser Schemas

The paper uses versioned prompt *families*. The row record stores the exact prompt text, prompt version, parser schema, and model-visible verifier state for every row; the templates specify the parser rules that turn model text into typed actions.

C.1 Mode Comparison

The four prompting modes evaluated in §5.1 (`plain`, `cot`, `operator`, CVL+RSS) hold the source task and parser fixed while varying the model-visible context, action format, and repair interaction.

Table 12: Prompting mode comparison. The modes share source task, parser schema, and validity rules while varying the context window, output format, and repair interaction.

Mode	Context shown	Output required	Search depth
`plain`	task statement only	repaired formulation, free form	1 turn
`cot`	task + CoT instruction	reasoning trace + repaired formulation	1 turn
`operator`	task + operator vocabulary	one structured operator + parameters	1 turn
CVL+RSS (CVL+RSS)	task + RSS fields + allowed probes + budget	action sequence with verifier deltas	multi-turn within budget

C.2 Fixed-Trace Inspection Prompt

Trace Inspection (§5.4) shows the model a completed verifier trace and asks for diagnostic labels in a compact JSON object:

Fixed-trace inspection output

```
{
"violated_invariant":      "<family>",
"decisive_step":         <integer>,
"next_verifier_action": "<operator>",
"reject_endpoint_rationale": <true|false>,
"counterfactual_prediction": "<label>",
```

```
"brief_reason":          "<one sentence>"
}
```

The parser accepts only declared labels for invariant families (`capacity`, `flow-balance`, `time-index`, `integrality`, `probability-norm`, `setup-link`, `skill-assign`) and verifier actions (`inspect_conflict`, `probe_family`, `repair_index`, `restore_integrality`, `repair_objective`, `normalize_scenarios`, `submit_repair`). Free-form explanations in `brief_reason` are stored with the row; scoring follows the parsed labels.

C.3 Live-Repair Output Schema

`plain` prompting receives only the task statement and asks for a repaired formulation in free form. `cot` adds a reasoning instruction in free form. `operator` exposes the action vocabulary and a compact verifier summary. The verifier-native pipeline (CVL+RSS) exposes the same action vocabulary plus the full RSS record, allowed CVL probes, and the run budget. In modes that require structured output the schema is:

Live-repair action schema (operator and CVL+RSS modes)

```
{
"action":        "inspect_conflict | probe_family | repair_index |
restore_integrality | repair_objective |
normalize_scenarios | submit_repair",
"target":        "<family>::<symbol>",
"after":         "<canonical token>",
"parameters":    { ... },
"expected_delta":  { "feasibility":  "+|-|0",
"rationality":  "+|-|0",
"objective":  "+|-|0" }
}
```

C.4 RSS Field Schema

The RSS compiler emits a record with five typed channels. The exact field schema, used both for the prompt body in CVL+RSS and for run serialisation, is:

RSS state record (typed five-channel input to CVL+RSS)

```
{
"status":     "OPTIMAL | INFEASIBLE | UNBOUNDED | TIMEOUT | ERROR",
"iis":        ["<family>[<index>]", ...],
"families":   ["capacity", "time-balance", ...],
"rationality":  {
"feasibility":  <bool>,
"balance":     "ok | violated",
"integrality":  "ok | relaxed",
"monotonicity":  "ok | violated",
"scenario_norm":  "ok | drift"
},
"recent":  [
{ "action":  "<op>", "target":  "<symbol>",
"delta":  { "feasibility":  "+", "rationality":  "0" } },
...
]
}
```

Empty IIS plus `status=OPTIMAL` is the only termination state that counts as a valid endpoint.

C.5 Parser and Validity Rules

The parser implements seven rules, applied in order: (i) the model output must be valid JSON; (ii) all keys must be from the declared schema; (iii) action labels must be from the declared vocabulary; (iv) target families must match `<family>::<symbol>` regex; (v) numeric parameters must lie in the allowed range; (vi) the expected-delta object must use only `+|-|0`; (vii) any remaining free text is stored with the row. A row that fails (i)–(vi) is recorded as a *parser failure*. Parser failure is a model-output failure and remains in the assigned-case denominator with strict repair score zero; the parser-valid companion score reports performance conditional on satisfying the declared schema.

Table 13: Capability and reliability accounting. The main strict metric uses the assigned-case denominator after construction and provider validity; parser failures remain model failures in the capability score.

Stage	Included rows	Use
Assigned cases	Source task, oracle, verifier replay, prompt version, and run root are present	Base experimental denominator
Provider-valid cases	Assigned cases without quota, transport, or nonretryable provider failure	Capability denominator
Parser-valid cases	Provider-valid cases whose model output satisfies the declared schema	Companion denominator
Strict success	Provider-valid cases with parser-valid canonical repair and verifier-accepted endpoint	Main strict repair score
Semantic success	Provider-valid cases with OR-equivalent repair under semantic acceptance rules	Legacy permissive diagnostic; see acceptance audit

C.6 Failed-Parse Example

The box shows a real model output that the live-repair parser rejects, paired with the rejection reason and row-level classification. The output passes JSON validation (rule i) but fails the action-vocabulary check (rule iii) and the typed `expected_delta` check (rule vi). The row is therefore stored as a parser failure, counted as a strict repair failure in the assigned capability denominator, and surfaced in the parser-failure ledger.

Rejected output (capacity-family repair)

```
{
"action":         "add_capacity_constraint",
"target":         "r3_t2",
"after":          "current_step",
"parameters":     { "rhs":  "adjust_to_demand" },
"expected_delta":  { "feasibility":  "yes",
"rationality":  "ok",
"objective":  "unknown" }
}
```

rejection: action label not in declared vocabulary (add_capacity_constraint ∉ {inspect_conflict, probe_family, repair_index, restore_integrality, repair_objective, normalize_scenarios, submit_repair}); target does not match the <family>::<symbol> regex; expected-delta values must be drawn from {+,−,0}, not free text; numeric parameter rhs is non-numeric.

classification: parser failure (rules iii, iv, v, vi). Original output preserved verbatim under conversations/<problem_id>.jsonl; row counted as strict repair failure and surfaced in the parser-failure ledger.

C.7 Failure-Handling Instruction

All prompts instruct the model to prefer a short structured answer over long reasoning text. If the model lacks enough information it must request an allowed probe before introducing new data or changing the OR semantics. Output-cap hits, malformed JSON, undeclared operators, and provider errors are recorded as parser or reliability events before any rerun, with the original output preserved verbatim with the row. Provider failures affect reliability accounting; malformed model outputs affect strict capability accounting.

C.8 Output handling in matched comparisons

The raw action schema above is unchanged for scoring. In the matched comparisons, CVL adds the normalization step defined in Appendix A.3 before execution; RSS-only does not. A case's normalized output is still judged by the same strict target and parameter checks. An explicit wrong value is never converted into the reference value. The benchmark-reference projector used in the separate interface diagnostic has a different purpose and is not this normalizer.

D Additional OR-Inspect-Bench Results

The appendix expands the per-model results behind Fig. 4 (§5.1). Each subsection connects a table to the repair behavior it measures. Table 14 contains all 23 models with strict and legacy semantic-v1 scores under `OR-cot` and CVL+RSS; four model rows are imputed reference rows and are marked with $\star$. Table 15 gives success counts, effective provider-valid denominators, and pooled raw-row counts for every displayed model/configuration pair. Table 16 adds the compact verifier-call and token diagnostics that vary across systems. Cost, wall-clock time, provider failures, and row-replacement bookkeeping are recorded in row-level records and omitted from capability tables because they are provider/accounting fields.

The result tables give a more granular mechanism reading. A lift is reported under matched source rows, parser rules, and validity policies, with verifier calls and tokens reported as diagnostics. Legacy semantic-v1 gaps record scorer disagreement; the acceptance audit in Appendix G.6 reports the corresponding adjudication evidence. Strict counts are unchanged by semantic-rule recalibration. The secondary tables show where verifier-state control helps most and where floor or saturation effects limit additional lift.

The starred Claude Opus, Claude Sonnet, Claude Haiku, and GLM-5 rows are calibration-based imputed reference values with nominal $n = 360$ per configuration. The remaining 19 model rows are directly measured runs and form the empirical cross-model comparison.

Table 17 supplies cluster-aware uncertainty for the isolated-state controller study. Its paired comparisons resample source-task clusters. The system matrix contains 19 measured model rows and four imputed reference rows; the matched controller studies report component effects on shared cases.

D.1 Run-Level Live Repair

Table 14 contains the complete matrix behind Fig. 4: it identifies where CVL+RSS converts one-shot failures into strict repairs or additional semantic-v1 acceptances, and where a model remains at the schema-following floor. Table 16 is the compact companion, reporting solver/checker/verifier calls and total tokens for every model in the CVL+RSS lane. The boundary is visible in the same table: verifier-call and token profiles show where a lift coincides with real interaction and where the row lies at a floor or saturation boundary. Near-zero lift rows mark schema-following failures outside the recoverable regime.

D.2 Trace Inspection

Trace Inspection evaluates completed verifier traces: the model is asked to identify the violated invariant, decisive step, next verifier action, endpoint-rationale rejection, and counterfactual implication. Agreement is computed against deterministic oracle labels derived from stored verifier states. The

Table 14: OR-INSPECT-BENCH live-repair results across 23 models. OR-cot is the one-shot chain-of-thought baseline; CVL+RSS is the full verifier-native pipeline. Lift = CVL+RSS strict RRR − OR-cot strict RRR. Scores use pooled provider-valid run records, so denominators vary when executions are repeated. Four imputed model rows are marked ⋆; per-cell success counts, valid denominators, and raw row counts appear in Table 15.

Model	OR-cot		CVL+RSS		Δ vs CoT
	Strict	Semantic-v1	Strict	Semantic-v1	
gpt-5.4	79.8	96.7	**93.3**	98.6	+13.5
gpt-5.2-chat	61.8	73.6	**91.4**	97.5	+29.6
gpt-4.1	61.4	74.4	**86.8**	94.9	+25.4
kimi-K2.5	22.2	25.8	**85.8**	93.9	+63.6
claude-opus-4-6⋆	67.2	87.5	**84.7**	96.4	+17.5
gemini-3.1-pro-preview	58.6	97.5	**81.3**	90.3	+22.7
gpt-5-mini	67.4	79.9	**79.4**	96.1	+12.0
gpt-5.4-mini	61.5	84.4	**79.1**	94.8	+17.6
qwen3.6-plus-preview	66.9	85.8	**78.9**	97.1	+12.0
claude-sonnet-4-6⋆	58.6	80.6	**76.9**	93.1	+18.3
gpt-4.1-mini	42.9	66.5	**76.8**	81.8	+33.9
DeepSeek-V3.2	36.5	69.3	**75.7**	93.1	+39.2
gemini-3.1-flash-lite-preview	72.3	92.1	**73.9**	93.0	+1.6
grok-3-mini	36.7	41.9	**69.7**	89.2	+33.0
bailian/glm-5⋆	50.6	80.8	**66.7**	88.6	+16.1
DeepSeek-R1-0528	34.6	74.0	**65.0**	86.0	+30.4
Qwen3-30B-A3B	18.8	29.6	**61.8**	79.8	+43.0
Llama-3.3-70B-Instruct	16.9	35.3	**58.3**	66.7	+41.4
gpt-5.4-nano	51.6	85.2	**58.2**	94.4	+6.6
gpt-5-nano	45.6	71.7	**55.6**	79.7	+10.0
claude-haiku-4_5⋆	38.3	67.5	**55.0**	75.0	+16.7
gpt-4o-mini	33.5	63.0	**45.6**	70.8	+12.1
qwen3-8b	1.6	2.7	**1.9**	2.7	+0.3

canonical pairwise set is the source of Table 18; larger diagnostic sweeps are retained in the row records, while the reported table uses the calibrated capability aggregation.

The main-text top-12 listing in Table 3 abridges the diagnostic ranking for space. The complete row set in Table 19 reports all evaluated systems with structural breadth, hard-action agreement, and measured token diagnostics. Effective denominators and row-replacement bookkeeping remain in the row records.

D.3 ENDPOINT BLIND SPOTS

Endpoint scoring is retained as a necessary diagnostic. Endpoint ties are the cases for which inspection is most informative. The canonical comparison has 72.0% endpoint ties and only 33.4% exact agreement with the operational oracle; adding rationality checks improves agreement to 45.4%. Inspection-style scoring follows the same trace-level evidence channel as the oracle and records trace-reading agreement.

D.4 PER-DIFFICULTY AND PER-FAMILY BEHAVIOUR

The DeepSeek-V3.2 anchor in Table 20 repeats the same strict/semantic quantities used in the main 23-model matrix. Lower-level verifier diagnostics remain in row-level records to avoid a second denominator for the same run. Qualitatively, the medium bucket is hardest in practice because it requires preserving an already-satisfied invariant family while repairing the violated one, so a wrong-but-feasible early action can mask the real fault. Easy cases admit a single localised repair, and hard cases require enough exploration for the CVL probe to determine the next safe repair.

Table 15: Strict success counts and denominators for the 23-model system matrix. Each cell reports successes / provider-valid rows (raw pooled rows). Provider-valid denominators exclude zero-token provider failures and include malformed outputs as strict failures. Repeated run records are pooled, not deduplicated by problem ID. All 46 displayed cells have zero parser-failure rows, so the rates match the stated scoring rule. A $\star$ marks an imputed model row, not a full empirical model evaluation.

Model	OR-cot: successes / valid (n_{raw})	CVL+RSS: successes / valid (n_{raw})
gpt-5.4	584/732 (732)	683/732 (732)
gpt-5.2-chat	445/720 (720)	658/720 (720)
gpt-4.1	448/730 (730)	625/720 (720)
kimi-K2.5	80/360 (360)	309/360 (360)
claude-opus-4-6*	242/360 (360)	305/360 (360)
gemini-3.1-pro-preview	211/360 (360)	317/390 (390)
gpt-5-mini	492/730 (730)	572/720 (720)
gpt-5.4-mini	450/732 (732)	579/732 (732)
qwen3.6-plus-preview	249/372 (372)	273/346 (346)
claude-sonnet-4-6*	211/360 (360)	277/360 (360)
gpt-4.1-mini	309/720 (720)	553/720 (720)
DeepSeek-V3.2	263/721 (724)	547/723 (727)
gemini-3.1-flash-lite-preview	276/382 (383)	275/372 (372)
grok-3-mini	132/360 (360)	251/360 (360)
bailian/glm-5*	182/360 (360)	240/360 (360)
DeepSeek-R1-0528	249/720 (720)	467/719 (720)
Qwen3-30B-A3B	70/372 (372)	230/372 (372)
Llama-3.3-70B-Instruct	61/360 (360)	210/360 (360)
gpt-5.4-nano	192/372 (373)	217/373 (374)
gpt-5-nano	164/360 (360)	200/360 (360)
claude-haiku-4_5*	138/360 (360)	198/360 (360)
gpt-4o-mini	178/532 (532)	183/401 (401)
qwen3-8b	6/373 (373)	7/373 (373)

Table 16: OR-INSPECT-BENCH live-repair score and diagnostics on the full-pipeline lane. Score columns come from replayed verifier outcomes; diagnostic columns are measured from the closest valid run record(s) for the same model and configuration. *Checks* is average solver/checker/verifier calls per case; *Tokens* is average total LLM tokens per case. Duration, gateway cost, and model-call counts remain in row-level records because they are provider- or protocol-dominated.

Model	Strict RRR	Semantic RRR	Δ vs CoT	Checks	Tokens
gpt-5.4	93.3	98.6	+13.5	7.6	6,490
gpt-5.2-chat	91.4	97.5	+29.6	7.8	7,369
gpt-4.1	86.8	94.9	+25.4	5.8	9,090
kimi-K2.5	85.8	93.9	+63.6	3.8	24,566
claude-opus-4-6	84.7	96.4	+17.5	3.9	4,938
gemini-3.1-pro-preview	81.3	90.3	+22.7	4.9	13,139
gpt-5-mini	79.4	96.1	+12.0	7.9	10,876
gpt-5.4-mini	79.1	94.8	+17.6	7.2	7,025
qwen3.6-plus-preview	78.9	97.1	+12.0	7.0	15,784
claude-sonnet-4-6	76.9	93.1	+18.3	4.5	4,292
gpt-4.1-mini	76.8	81.8	+33.9	6.6	10,528
DeepSeek-V3.2	75.7	93.1	+39.2	7.3	10,100
gemini-3.1-flash-lite-preview	73.9	93.0	+1.6	5.9	7,970
grok-3-mini	69.7	89.2	+33.0	3.9	10,956
bailian/glm-5	66.7	88.6	+16.1	3.7	6,005
DeepSeek-R1-0528	65.0	86.0	+30.4	6.3	24,622
Qwen3-30B-A3B	61.8	79.8	+43.0	5.4	14,909
Llama-3.3-70B-Instruct	58.3	66.7	+41.4	3.1	19,696
gpt-5.4-nano	58.2	94.4	+6.6	7.8	6,994
gpt-5-nano	55.6	79.7	+10.0	8.4	17,337
claude-haiku-4_5	55.0	75.0	+16.7	2.3	4,073
gpt-4o-mini	45.6	70.8	+12.1	9.4	5,038
qwen3-8b	1.9	2.7	+0.3	5.2	10,840

Each OR-INSPECT-BENCH case is annotated with one of the eight process families listed in Table 10. We aggregate per-family RR/RRR for every model run and use those aggregates to drive the mechanism interpretation in §5.1. The two families where CVL+RSS contributes most over `OR-cot` are *feasible-rationality split* and *backslide*: both are cases an endpoint checker can silently accept, and both are cases where the CVL rationality channel produces the verifier delta that drives the next

Table 17: Sensitivity of the matched CVL minus RSS margin to clustering. All deltas and intervals are in percentage points, from 2,000 paired draws. The eight process-family units provide a coarse sensitivity analysis, not fine-grained template or crossed-effects inference.

Model	Δ	120 source-task clusters	8 process-family clusters
Qwen3-30B-A3B	+10.8	[+5.6,+16.7]	[+5.2,+16.0]
GPT-5.4-nano	+17.2	[+12.5,+21.7]	[+13.6,+21.2]
GPT-5.4-mini	+3.3	[+0.6,+6.4]	[−0.7,+7.1]
GPT-4.1-mini	+3.1	[0.0,+6.7]	[0.0,+5.8]
Kimi-K2.5	+0.8	[−1.4,+3.1]	[−0.7,+2.4]

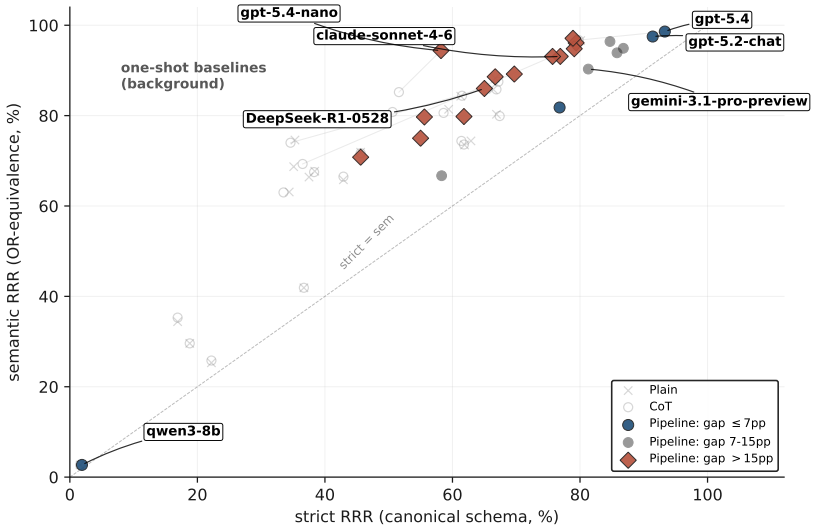


Figure 6: STRICT_RRR vs. legacy semantic-v1 across OR-INSPECT-BENCH modes. Light grey marks one-shot baselines; foreground points are CVL+RSS runs. Distance from the diagonal shows strict/semantic-v1 disagreement; it does not certify that a rejected strict repair is semantically correct.

repair. The DeepSeek difficulty anchor supplies a compact family-level view, while the measured model rows provide the cross-model evidence and family aggregates are retained in the row records.

D.5 REPRESENTATIVE TRAJECTORY VIGNETTES

A fixed-trace inspection row contains a frozen trace, the violated invariant, the decisive step, the next verifier action, an endpoint-rationale rejection label, and a counterfactual prediction label. A live-repair row contains the source task, scaffold state, parsed action sequence, verifier deltas, final endpoint status, rationality status, and run-status fields. These row objects illustrate how the three paper objects appear at row level: RSS exposes the repair state, CVL acts on verifier deltas, and OR-INSPECT-BENCH scores the full repair trajectory.

Fixed-trace diagnostic row (trace inspection)

```
source:  supply_planning / capacity-time balance
visible trace:
step 0:  solver = INFEASIBLE, iis = {cap[t=2,r=3]}
step 1:  action = relax_integrality, verifier delta = (0, 0, 0)
step 2:  action = add_capacity(r=3,t=2), solver = OPTIMAL
step 3:  rationality.balance = ok, objective = 14820
diagnostic query:  identify violated invariant, decisive step, and
next verifier action.
oracle labels:  violated_invariant = capacity; decisive_step = 2;
```

Table 18: OR-INSPECT-BENCH protocol agreement against the operational oracle on the canonical pairwise set (N=500). *Exact agreement %* is the per-row exact match with the oracle; *Tie rate %* is the fraction of tied comparisons. Endpoint scoring ties most cases (72%) and matches the oracle about one-third of the time; adding rationality probes resolves part of those ties and lifts agreement to 45%. Inspection-style scoring uses the same trace-level evidence channel as the oracle and records trace-reading agreement.

Protocol	Exact agreement	Tie rate
Endpoint-only	33.4%	72.0%
Endpoint+rationality	**45.4%**	**60.0%**

Table 19: Full TRACEJUDGE fixed-trace diagnostic results after row-level problem de-duplication. *Diagnostic* is the six-part exact judgement used in the main table. *Structural* averages the four easier structural fields and diagnoses saturation. *Hard-action* averages next action, endpoint-rationale rejection, and counterfactual prediction. Duration, gateway cost, and model-call counts remain in row-level run records.

System	Diagnostic	Structural	Hard-action	Tokens
gpt-5.4	80.0	91.9	70.1	923
gpt-5.2-chat	79.9	91.1	67.4	1,013
claude-opus-4-6	75.0	95.1	55.9	1,140
gemini-3.1-pro-preview	71.7	90.6	44.1	1,813
gpt-5-nano	70.3	84.6	50.7	2,969
gpt-4.1-mini	69.3	79.2	58.7	867
kimi-K2.5	68.6	91.1	45.5	2,897
gpt-5.4-nano	66.3	73.4	52.4	1,002
gpt-5.4-mini	65.8	85.4	40.6	899
grok-3-mini	64.4	91.4	33.7	792
DeepSeek-V3.2	62.0	76.0	51.7	981
gpt-4.1	59.9	89.8	27.4	864
claude-sonnet-4-6	57.6	80.7	32.6	1,061
Qwen3-30B-A3B	57.3	81.2	26.4	1,361
gpt-5-mini	55.7	74.2	29.5	1,994
gpt-4o-mini	54.7	60.7	54.2	823
claude-haiku-4_5	54.0	69.0	27.4	1,174
DeepSeek-R1-0528	52.1	71.9	29.9	3,363
qwen3-8b	45.1	66.9	16.0	1,214
qwen3.6-plus-preview	44.4	53.9	32.3	1,842
gemini-3.1-flash-lite-preview	44.3	66.1	9.7	929
Llama-3.3-70B-Instruct	43.1	64.1	18.8	813

next_verifier_action = submit_repair; reject_endpoint_rationale = false.
scoring: exact matches enter TraceJudge sub-scores; typed labels determine the score, with the free-text reason stored as row text.

Live-repair row (full CVL+RSS lane)

source task: facility assignment with capacity and compatibility families.
RSS state: status = OPTIMAL; iis = ∅; families = {capacity, compatibility}; rationality.compatibility = violated; recent = {relax_capacity: feasibility +, rationality −}.
CVL probe: probe_family(compatibility::facility_A) predicts feasibility 0, rationality +.
accepted action:
action = submit_repair; target = compatibility::facility_A;
after = add_compatibility(customer_i, facility_A);
expected_delta = {feasibility: 0, rationality: +, objective: 0}.
verifier outcome: solver = OPTIMAL; rationality.compatibility = ok; strict RRR = 1; semantic RRR = 1.

Table 20: DeepSeek-V3.2 OR-INSPECT-BENCH anchor on the nominal 360-case live-repair slice. The pooled full-pipeline score is 547/723 clean execution rows (75.7%); four of 727 raw rows have zero tokens. Values match the main 23-model matrix.

Metric	Value
Strict RRR	75.7%
Semantic RRR	93.1%
Lift vs. CoT	+39.2 pp

Table 21: Prompt and control modes used for live-repair ablations. The rows share source tasks, parser rules, validity gates, and repair budgets where applicable; they vary the feedback representation and control interface.

Mode	Model-visible structure	Capability measured
CoT prompting	Plain prompt plus reasoning instruction	Whether free-form reasoning alone improves repair decisions
Raw MT feedback	Multi-turn solver/domain feedback rendered as text	Whether repeated verifier feedback helps without typed state control
RSS MT feedback	Multi-turn feedback compiled into RSS fields	Whether typed verifier state helps beyond raw feedback
Verifier-native pipeline	RSS, CVL probes, operator retrieval, parser, verifier execution	Full verifier-state interface with controller selection and replayed verifier checks

These examples illustrate three recurring phenomena in the aggregate results (Table 14, Fig. 6, Table 20, Table 22). First, endpoint recovery can be false recovery: the solver status becomes `OPTIMAL` after a relaxation, while a rationality flag remains violated. OR-INSPECT-BENCH records this as endpoint acceptance without rational repair. Second, repair can backslide: a candidate clears the current IIS family while breaking a previously satisfied balance, linking, or quota family. CVL uses the verifier delta to preserve the previous frontier state in such rows; this phenomenon is reflected in the difficulty/family view of Table 20. Third, state exposure and controller selection can separate across model families. The Qwen anchor gains strongly from RSS exposure, but the full controller can lose strict successes. A permissive semantic acceptance does not establish that such a failure is merely formatting; the audit includes explicit wrong objective directions. This is why the ablation reports staged modes without imposing a monotone ordering (Table 21, Table 22).

D.6 ABLATION INTERPRETATION

The complete screen in Table 22 distinguishes raw feedback, typed RSS, and full-controller knockouts. Typed state improves on raw feedback across the three anchors; adding the full controller is model-dependent. Both no-memory and no-probes controls recover Qwen successes relative to full control, but neither establishes that probes are the exclusive cause. The stored action/probe traces do not support that causal attribution. The isolated-state CVL comparisons and their normalization boundary are reported separately in Appendix G.

E EXTERNAL PUBLIC-BENCHMARK TRANSFER

Public benchmarks test whether the same run protocol produces accepted final formulations outside the OR-INSPECT-BENCH trajectory interface. Each row keeps the source benchmark's scoring rule and the denominator used in our transfer run: the LEAN-LLM-OPT 101-task subset, NL4Opt's 245 natural-language LP tasks, a 100-row MAMO-Easy subset, a 203-row filtered MAMO-Complex subset, and 100 IndustryOR industrial modelling tasks. A row is credited when the final formulation is accepted under that benchmark's native parser, solver, and invalid-output policy.

The literature baseline cells below are source-reported reference values, not reruns under our prompts, harness, or invalid-output policy. Differences across these cells do not identify a matched controller

Table 22: Configuration screen (May 2026): strict repair percentages on 360 matched cases per model. The no-probes row does *not* include the normalizer used by CVL in the matched study. These absolute scores are not compared across windows with the isolated-state study. The full configuration is not uniformly best; the RSS baseline and all screened configurations are retained.

Configuration	GPT-5.4-mini	DeepSeek-V3.2	Qwen3-30B-A3B
CoT	49.7	53.1	11.1
Operator prompt	48.6	51.7	11.9
Raw multi-turn	76.7	68.9	52.2
RSS-only multi-turn	83.3	78.1	80.8
Full CVL, no memory	86.4	76.4	75.3
Full CVL, no probes	89.2	80.8	76.7
Full CVL	84.7	78.3	61.4

effect. They are retained to situate the native-protocol runs; Table 2 is the direct RSS versus CVL comparison.

E.1 Public Benchmark Protocols

The public suite differs in surface language, ground-truth format, and denominator. Rows are averaged only within identical benchmark/source/model/config groups, and repeat values remain in the row records. Table 23 reports the per-system MA profile across the five public benchmarks alongside a cohort of published baselines spanning the OR-LLM literature (Huang et al., 2025a; Xiao et al., 2024; AhmadiTeshnizi et al., 2024; Liang et al., 2026) and zero-shot general-purpose models; the column ranges form a *published reference band*. The row records store benchmark version, denominator, baseline source, model/provider alias, prompt mode, repeat policy, invalid-row policy, and native metric.

Table 23: External public-benchmark transfer with the full published reference cohort. Numbers are modeling accuracy (MA, %) under each benchmark's native scoring protocol. Superscripts mark the source family for each baseline; em-dash = not reported by that source; bold = this paper.

Group	System	NL4Opt	MAMO-Easy	MAMO-Complex	IndustryOR	LEAN-LLM-OPT-101
This work						
	Ours (DeepSeek-V3.2 + CVL+RSS)^{T}	**98.4**	**95.0**	**85.7**	**80.0**	**88.1**
Agentic LEAN pipelines						
	LEAN-GPT-4.1^{L}	97.1	93.5	74.9	65.0	85.1
	LEAN-OSS-20B^{L}	93.9	92.2	56.7	59.0	80.2
	LEAN-GPT-4.1 (agnostic)^{L}	—	—	—	—	54.5
	LEAN-OSS-20B (agnostic)^{L}	—	—	—	—	50.5
Zero-shot / raw closed models						
	Gemini-3-Pro^{L}	87.7	90.3	70.4	65.0	52.5
	GPT-5.2^{L}	82.0	92.8	67.5	56.0	40.6
	GPT-5^{L}	78.8	92.3	54.4	58.0	30.7
	GPT-4.1^{L}	72.7	89.6	57.1	54.0	14.9
	gpt-oss-20B^{L}	50.2	68.9	29.1	42.0	25.7
	GPT-4 (standard prompting)^{O}	47.3	66.5	14.6	28.0	—
OR-LLM trained models						
	ORLM-Mistral-7B^{O}	84.4	81.4	32.0	27.0	—
	ORLM-LLaMA-3-8B^{O}	85.7	82.3	37.4	38.0	—
	ORLM-DeepSeek-Math-7B^{O}	86.5	82.2	37.9	33.0	—
Multi-agent / CoT systems						
	GPT-3.5 + Chain-of-Experts^{C,O}	58.9	—	—	—	—
	Chain-of-Experts (best)^{C}	64.2	—	—	—	—
	OptiMUS (GPT-4)^{M}	78.8	77.2	43.6	31.0	5.0

^{T} This paper. ^{L} LEAN-LLM-OPT reference tables and released result files (Liang et al., 2026). ^{O} ORLM Table 1 and associated public-benchmark reporting (Huang et al., 2025a). ^{C} Chain-of-Experts / OR-LLM-Agent reported NL4Opt rows (Xiao et al., 2024). ^{M} OptiMUS reported public-benchmark rows (AhmadiTeshnizi et al., 2024).

Table 24: External public-benchmark native score with compact token diagnostics. Each benchmark uses the denominator stated in Table 23; repeat values and full runtime diagnostics stay in row-level records. Token columns are measured from the same run root used for the reported MA whenever available.

Benchmark	Native MA	Tokens	Tokens per call
NL4Opt	98.4	1,640	740
MAMO-Easy	95.0	2,891	1,041
MAMO-Complex	85.7	9,099	2,090
IndustryOR	80.0	6,330	1,978
LEAN-LLM-OPT-101	88.1	26,562	3,120

Table 24 reports token-use diagnostics next to the native metric; denominators are already stated in Table 23 and the row record. Full call counts, durations, provider-cost fields, and timeout flags remain in the row-level records. These fields are omitted from the table when they are protocol constants, all-zero columns, or provider-accounting artefacts.

The public rows report accepted-formulation accuracy under each external protocol. They measure transfer across benchmark formats; OR-INSPECT-BENCH remains the main trajectory-repair experiment. The harder public denominators stress formulation and verifier-state transfer, while saturated NL4Opt/MAMO-Easy cells check preservation on established endpoint tasks.

The suite includes both high-accuracy and difficult formulation settings. The different source protocols prevent interpreting small cross-paper score differences as statistically resolved effects, or attributing them to a single controller component. The matched OR-Inspect study is the evidence for state and controller attribution; these external rows document execution under established modeling-accuracy protocols.

Provider failures and superseded runs remain in row-level records and separated from model-capability scores. Timeout rows are counted according to the benchmark's native rule if that rule requires it; otherwise they remain reliability diagnostics. Capability denominators follow the benchmark's published protocol. The reported mode names are Plain prompting, CoT prompting, Operator prompting, and Verifier-native pipeline; historical adapter nicknames and internal run labels are row-record metadata only.

Table 25: Feedback roles in related approaches. These are representative roles, not exclusive capabilities. Typed tools, closed-loop repair, and search predate this work; our study specifies an OR interface and measures its components.

Paradigm	Feedback signal	How feedback is used	Evaluation object
OR-LLM endpoint benchmarks	Solver result	Terminal grading	Final formulation
Reflection/tool agents	Text or structured tool output	Reflection or search	Task and process outcomes
RLVR / process rewards	Verifiable or learned reward	Training or learned step judging	Outcome or process score
Solver-in-the-loop OR repair	Solver feedback	IIS-guided repair and validation	Feasibility and rational recovery
CVL+RSS (ours)	**Exact verifier-state delta**	**Training-free control over typed state**	**Repair trajectory**

F Extended Discussion of Related Work

The main text positions the paper against endpoint OR benchmarks, general tool-use agents, process-reward methods, and solver-feedback repair. The expanded comparison traces how each area uses feedback: as an endpoint grade, a textual critique, a training signal, or a verifier state transition.

F.1 OR-LLM Modeling and Endpoint Benchmarks

Public OR-LLM benchmarks measure end-to-end translation accuracy under a one-shot or multi-shot prompt: NL4Opt (Ramamonjison et al., 2023) established the natural-language-to-LP task at NeurIPS 2022; MAMO (Huang et al., 2025b) extended this with easy and complex LP instances paired with solver-correctness checks; LEAN-LLM-OPT (Liang et al., 2026) reports a published reference suite across NL4Opt, MAMO, IndustryOR, and a LEAN-augmented modeling pipeline; ORLM (Huang et al., 2025a) releases a customisable training framework with fine-tuned modeler weights; OptMATH (Lu et al., 2025) synthesises bidirectional NL/LP data at scale; and DP-Bench (Zhou et al., 2025) probes dynamic-programming formulation, where recent models reach roughly $\sim 60\%$ accuracy. On the system side, multi-agent and planning frameworks include Chain-of-Experts (Xiao et al., 2024), Optimus (AhmadiTeshnizi et al., 2024), and OptimAI (Thind et al., 2025); Bertsimas & Margaritis (2024) extends LLM-based modeling to robust and adaptive optimization. All of these treat the formulation step as the unit of evaluation: a model emits a complete formulation, the formulation is solved, and the endpoint correctness is graded against a reference. OR-Inspect-Bench keeps the formulation step as a baseline mode (`plain`, `cot`, `operator`) but adds a *repair-trajectory* mode under exact verifier feedback, where credit is assigned to the sequence of interventions.

F.2 OR Repair and Solver Feedback

OptiRepair studies IIS-guided feasibility repair together with five supply-chain rationality checks over 976 problems and 22 API models (Ao et al., 2026a). ORLoopBench casts solver-in-the-loop repair as an MDP and evaluates 5,362 LP/MILP repair instances, with complementary behavioral-rationality tasks (Ao et al., 2026b). These systems provide important reference points, while the present study focuses on a different experimental axis: the contract and composition of inference-time repair interfaces. Specifically, RSS exposes five typed verifier channels; matched interventions separate structured feedback and output normalization from additional controller configuration; and OR-Inspect-Bench combines fixed-trace diagnosis with live repair under shared task and scoring definitions. The resulting analyses quantify component effects and model-conditional control margins across distinct verifier-mediated repair configurations.

F.3 Self-Correction in General LLM Reasoning

General self-correction literature spans verbal-reflection methods (Yao et al., 2023b; Shinn et al., 2023; Madaan et al., 2023; Gou et al., 2024; Zelikman et al., 2022), search augmentation (Yao et al., 2023a; Zhou et al., 2024), and explicit RL training for self-correction (Kumar et al., 2025). CorrectBench (Tie et al., 2025) studies cross-domain self-correction across commonsense reasoning, mathematical reasoning, and code generation, and reports that more than half of all errors stem from *correction failures* after the initial mistakes. Our setting focuses on deterministic OR verifier feedback. SWE-bench (Jimenez et al., 2024) evaluates code repair through unit-test feedback, using tests and repository-level tasks. Our OR adapters expose domain-specific diagnostics—status, IIS, slack, and rationality checks—under a controlled perturbation protocol. Our CVL loop treats the verifier delta Δ_t as the control signal, with self-judged rationales kept outside the controller.

F.4 RLVR, Process Reward, and Verifier-Based Systems

RL with verifiable rewards (DeepSeek-AI, 2025; Lambert et al., 2024; Shao et al., 2024; Yu et al., 2025) trains policies from outcome signals; Yue et al. (2025) argues this mostly sharpens base-model capabilities with limited expansion of the reasoning boundary. Process-reward models supervise intermediate steps (Wang et al., 2024; Lightman et al., 2024; Luo et al., 2024; Zhang et al., 2024), and Zhang et al. (2026b) extends process-style verification to multimodal reasoning. Closed-loop world models (Zhang et al., 2026a) predict counterfactual rollouts to score candidate plans. In OR, exact solver and domain verifiers provide a deterministic process signal over typed state; CVL+RSS consume this signal directly at inference time as a training-free controller.

F.5 Behavioral Rationality and OM Decision Biases

Operations Management has a long line of empirical work documenting human decision biases under uncertainty: pull-to-center in newsvendor ordering (Schweitzer & Cachon, 2000; Bolton & Katok, 2008), system-neglect in demand forecasting (Kremer et al., 2011), and bias persistence under information-rich conditions. Recent work asks whether LLMs inherit these biases when used as inventory agents (Zhao et al., 2025). Our *rationality* channel encodes one slice of this concern as deterministic verifier-state checks (balance, integrality, monotonicity, scenario-normalisation), so that a formulation with optimal endpoint status but violated quota or service semantics is correctly flagged in the live-repair lane. This narrows the gap between formal feasibility and operational correctness, which is the gap a behavioral-OM benchmark targets through human-decision-style probes.

F.6 Test-Time Scaling and Inference Compute

Inference-compute scaling literature studies how extra generation/sampling/verification trades off against model capacity (Snell et al., 2024; Cobbe et al., 2021). CVL sits in this family, with a different accounting rule: additional inference rounds are recorded against their observed verifier deltas. Extra calls affect capability credit only when they move the RSS signature toward the targeted repair. We report token, call, and wall-clock budgets in Appendix D so that the inference-compute axis is traceable per row.

F.7 Benchmark Reliability and Long-Horizon Agent Evaluation

The reliability of small-N benchmark rankings is itself a research problem. Yauney et al. (2026) shows that small micro-benchmarks can be unstable and that substantially larger subsets, sometimes around 250 examples, may be needed for reliable pairwise ranking; we report the full 360-row OR-Inspect-Bench denominator as the main slice and pair it with a 50-canonical anchor for protocol diagnosis (App. D). Broader agent-evaluation work includes TheAgentCompany (Xu et al., 2025), which probes LLM agents on long-horizon real-world tasks, and MLRC-Bench (Zhang et al., 2025), which critiques LLM-as-judge protocols for ML-research challenges. For OR repair, OR-Inspect-Bench provides a complementary trajectory-level evaluation in which deterministic verifier evidence supports attribution of progress to specific repair decisions. Wu et al. (2025) introduces multi-turn-aware reward design for collaborative agent tasks. CVL makes this principle operational for OR repair: typed verifier-state transitions supply the search reward, linking each

controller decision to measured progress on solver and domain checks. The isolated-state evaluation disables online experience writes to avoid cross-run dependence.

G Controlled Evaluation and Acceptance Audit

This section separates the evidence for typed feedback, controller configuration, output handling, and evaluation reliability. All repair rates below are strict unless a semantic-rule version is explicitly named. None of the tables selects the better controller separately for each test problem.

G.1 Configuration and evaluation scope

The broad study reports the full CVL+RSS system in a 23-row matrix, with 19 directly measured model snapshots and four marked imputed reference rows. The seven-configuration screen in Table 22 compares prompting, raw feedback, typed feedback, and controller knockouts on three anchors. Its May 2026 runs are distinct from the July 2026 isolated-state controller study. These are separate campaign windows; within each reported controller pair, the problem IDs, provider/model endpoint, and scoring rule are matched. The output-interface diagnostic is a third, separately matched campaign.

Table 26: Configuration differences relevant to attribution. "Normalizer" is the reference-free repair-completion rule of Appendix A.3; "projector" instead denotes the benchmark-specific diagnostic mapping. The unqualified CVL row is the probe-free configuration used in matched comparisons.

Condition	Feedback / interaction	Additional control	Output handling
CoT	One-shot prompt	No search	Canonical schema
Operator	One-shot operator schema	No search	Canonical schema
Raw MT	Raw verifier text, multi-turn	Feedback loop	Canonical schema
RSS-only	Typed feedback, multi-turn	Feedback loop	Canonical schema
Full CVL	Typed feedback	Search, probes, operator retrieval	Canonical schema
CVL	Typed feedback	Search without probes; retrieval eligible	Reference-free normalizer
Self-consistency	Typed feedback	CVL plus three candidates	Same normalizer
Interface diagnostic	Same typed feedback as RSS	Same nominal interaction budget	Free-form output plus projector

The isolated-state protocol uses an empty run-local experience store without bootstrap trajectories and disables online writes. It prevents one experiment's outputs from becoming another experiment's retrieved experience. Fixed operator knowledge and the eligibility of retrieval calls are separate from online experience accumulation. Cross-run-sensitive exploratory executions are excluded from the matched comparisons. A resumed execution retains completed unique problem/model/configuration records rather than counting them twice.

Qwen within-run comparison. The RSS, CVL, and three-candidate self-consistency conditions run in problem-major order with concurrency one. All three contain 360 unique, provider-valid rows. The model endpoint is `Qwen3-30B-A3B`, with thinking disabled for this evaluation. The complete comparison spans July 24–25, 2026, including an interruption and resume, with completed unique problem/configuration records retained once. Table 27 reports usage over the full slice. The additional controller improves the strict count with higher token use; self-consistency uses more calls and yields a lower strict count than CVL.

Each of the RSS and CVL conditions records 361 LLM calls: 359 cases use one call and one case uses two. Verifier-path lengths differ: for RSS, 24/120/215/1 cases use 4/5/6/8 solver calls; for CVL, 330/29/0/1 cases use 4/5/6/10 solver calls. These counts separate LLM decision turns from solver/verifier interaction depth.

Fresh matched normalizer comparison. A separate September 25, 2026 run compares RSS-only, RSS plus the reference-free emission normalizer, and CVL-r2 on the same ordered 360-case manifest. All arms have 360 model-call and token records and zero parser failures. The strict-RRR counts are

Table 27: Same-run Qwen comparison on 360 cases. Calls and tokens are totals, including recorded calls beyond a single response per row. Provider dollar-cost fields are unavailable for this run.

Condition	Strict successes	Strict (%)	LLM calls	Tokens
RSS-only	196	54.4	361	898,153
CVL	235	65.3	361	1,077,595
Three-candidate self-consistency	226	62.8	1,112	3,302,045

110, 154, and 165, respectively; paired source-task and family-cluster intervals for both contrasts are reported in §5.2. Total LLM calls are 360, 360, and 370, with the CVL-r2 calls distributed as 351 one-call, 8 two-call, and 1 three-call cases. Solver-call distributions are 24/120/216 cases at 4/5/6 calls for each RSS arm, and 14/261/62/6/17 cases at 3/4/5/6/7 calls for CVL-r2. Total token counts are 709,188, 708,860, and 776,343. This run directly estimates the normalizer contrast on matched cases and records the realized interaction depth alongside the repair outcomes. Paired 95% intervals use 20,000 percentile bootstrap replicates (seed 20260925), resampling the 120 source-task clusters and, separately, the eight process-family clusters.

The other model anchors in Table 2 are matched within their own evaluation windows. Their RSS / CVL success counts are 116/178 for GPT-5.4-nano, 276/288 for GPT-5.4-mini, 318/329 for GPT-4.1-mini, 346/349 for Kimi-K2.5, and 261/257 for DeepSeek-V3.2, each out of 360. The mixed signs and uncertainty characterize when additional control changes outcomes across the evaluated model anchors.

G.2 Paired uncertainty

Let $y_{ic} \in \{0, 1\}$ be the strict outcome on problem i under condition c. For two matched conditions, the target is

$$\widehat{d} = \frac{100}{N} \sum_{i=1}^{N} (y_{i,\text{target}} - y_{i,\text{base}}) \quad \text{percentage points.}$$

The main slice has 120 source tasks, each with three hardened variants. In each of 2,000 bootstrap draws (seed 42), sample 120 source-task clusters with replacement and retain all rows and both conditions in each sampled cluster. Recompute the row-weighted paired difference and report its 2.5th and 97.5th percentiles. This preserves dependence among variants of the same source task. Table 17 gives the intervals for all six matched-controller anchors, including unresolved and negative effects.

A second analysis resamples the eight recorded case/process families, which provide a coarser sensitivity grouping than source-task clusters. The intervals describe sampling variation in this benchmark under the recorded executions; they do not include uncertainty from model-endpoint drift, repeated configuration exploration, or a future deployment population.

G.3 State, ranking, and interaction controls

The one-shot operator-schema control changes strict accuracy by $-1.1/-1.4/+0.8$ pp relative to CoT for GPT-5.4-mini/DeepSeek-V3.2/Qwen3-30B. By contrast, raw multi-turn feedback followed by typed RSS gives much larger state-exposure gains in Table 22. The operator control holds the parser fixed; it does not itself measure parser sensitivity.

For Qwen, random action selection obtains 103/360 strict successes (28.6%), compared with 235/360 (65.3%) for lexicographic selection under the reported candidate-generation condition. This tests whether the selection rule matters; it is not an additive decomposition of independent component effects. On the 144-case exact subset where retrieval is not invoked, RSS and CVL succeed on 95 and 121 cases (66.0% and 84.0%). This slice reports the matched outcomes on the no-retrieval branch separately.

Qwen probe-by-canonicalization factorial. The same-window 2×2 comparison holds the 360 Qwen problem IDs fixed while varying probe budget and canonical-emission correction within

the CVL configuration family. Table 28 reports the four strict success rates. Canonical-emission correction increases strict success by 14.4 pp with probes enabled (source-task 95% CI $[8.3, 20.6]$; family CI $[7.4, 21.4]$) and by 20.6 pp with probes disabled (source-task CI $[14.4, 26.7]$; family CI $[15.6, 25.5]$). Both contrasts are positive under the two cluster definitions. Probe-budget contrasts and their interaction include zero under both definitions. This factorial isolates the correction effect within these CVL variants; the separate three-arm study above directly compares RSS with and without the normalizer and measures the additional CVL-r2 package margin.

Table 28: Same-window Qwen probe-by-canonicalization factorial on the same 360 problem IDs. Strict repair success counts and rates are shown. The four cells are CVL configuration variants; the separate three-arm study above directly compares RSS with and without the normalizer.

Probe budget	Correction off	Correction on	Correction effect
On	182/360 (50.6%)	234/360 (65.0%)	+14.4 pp
Off	169/360 (46.9%)	243/360 (67.5%)	+20.6 pp

Table 29: Usage in the three-anchor full-controller screen. Each entry is a per-case mean on that screen, not for the probe-free matched comparisons.

	Verifier calls		Tokens	
Model	RSS	Full CVL	RSS	Full CVL
GPT-5.4-mini	5.53	3.91	1813.7	1868.9
DeepSeek-V3.2	5.53	3.51	2052.4	2855.5
Qwen3-30B-A3B	5.59	4.14	2410.5	2977.6

The full controller uses fewer verifier calls but more tokens in this screen (Table 29). Thus more verifier access does not explain its outcomes, while equal total computation is not established. Probe-free CVL has its own accounting in Table 27; the full-controller budget cannot be transferred to it. Both no-probe and no-memory knockouts recover some Qwen successes relative to full control. The stored traces do not identify an exclusive probe-related cause, and permissive semantic acceptance cannot establish that all lost strict successes were merely formatting errors.

G.4 Held-out compositions and feedback noise

A frozen 48-case slice uses new compound perturbation orders and compositions, with 16 easy, 16 medium, and 16 hard cases across eight process families. It is materialized after the controller is fixed. Source tasks, primitive error types, and action grammar are reused: this is composition transfer, not unseen-domain or unseen-primitive transfer. Every case passes source, perturbation-oracle, parser, and validity replay. These four checks establish construction consistency; they are not an independent expert certification.

Table 30: Held-out compositions: strict success counts out of 48. Wins and losses are paired RSS–CVL flips, not independent Bernoulli trials on their net difference. We report counts and point estimates for this small slice.

Model	CoT	RSS	CVL	Wins	Losses	Margin (pp)
GPT-5.4-mini	18	30	36	7	1	+12.5
DeepSeek-V3.2	14	26	26	0	0	0.0
Qwen3-30B-A3B	5	20	31	15	4	+22.9
Kimi-K2.5	23	48	48	0	0	0.0

A separate Qwen stress test applies the configured 25% corruption to model-visible checker observations, with the final scoring oracle unchanged. RSS succeeds on 189/360 cases (52.5%) and CVL on 231/360 (64.2%). This single model and noise setting is an exploratory robustness check; it does not establish robustness to arbitrary noisy, delayed, or missing verifiers.

G.5 Output-interface sensitivity and portability

The output-interface comparison fixes typed RSS input, the 360 problem IDs, and the nominal multi-turn budget within a separate Qwen execution. The canonical condition requires a structured repair. The free-form condition instead passes the model's stated repair intent through the benchmark's output projector, which maps recognized intent to reference actions.

Table 31: Output-interface diagnostic. The projector uses benchmark reference actions and is not the reference-free CVL normalizer. Semantic-v1 is shown only to identify the recorded scoring rule, not as validated correctness.

Condition	Strict / 360	Semantic-v1 / 360	LLM calls	Tokens
Canonical RSS output	181	359	360	898,048
Free-form plus projector	358	358	360	585,125

The strict difference is +49.2 pp, with source-task/family intervals $[42.2, 55.8]/[42.9, 56.9]$; the recorded semantic-v1 difference is approximately -0.3 pp. This is evidence that canonical-emission requirements materially change the measured score. The projector supplies benchmark-specific action completion, so the 99.4% strict rate is not an independently deployable repair result. In particular, unchanged permissive semantic-v1 scores are not proof of unchanged true OR correctness.

An additional generic ReAct-style lane uses raw task/checker information and free-form repair output without RSS or CVL ranking. It reaches 308/360 strict successes (85.6%) with 2.375 mean LLM calls, 6.7 mean verifier calls, and approximately 4,447 tokens per case. Its output is handled by the same benchmark-specific free-form projection mechanism. This demonstrates that another input/control interface can exercise the benchmark, but does not isolate the controller, provide an off-the-shelf commercial-agent baseline, or establish superiority under matched computation.

G.6 Acceptance audit and semantic-rule calibration

The audit uses 125 selected cards: 93 Qwen disagreements, 15 analogous non-Qwen disagreements, 12 positive controls, and five missing-payload negative controls. It deliberately over-samples cases accepted by semantic-v1 but rejected by strict scoring; it is not a random sample of all benchmark decisions. The adjudicator is `claude-opus-4-5`, from a different model provider than the OpenAI, DeepSeek, and Alibaba agents being inspected.

The judge receives the audit instructions and one card, without the producer's identity, stratum, saved verdict, or answer key. Judgments are saved before the key is joined to the results. There are 120 definite judgments (16 accepts, 104 rejects) and five unsure judgments, all on missing-payload controls. Table 32 compares rules on the definite subset.

Table 32: Agreement with the blinded model adjudicator on the selected definite cards. Semantic-v2 is calibrated and evaluated on the same cards; rates are descriptive agreement on this selected audit set.

Rule	Agreement	Rate	Interpretation
Strict	112/120	93.3%	Canonical repair score; unchanged by calibration
Semantic-v1	16/120	13.3%	Permissive concept matching over-accepts in this selected set
Semantic-v2	108/120	90.0%	Adds objective-direction restoration; same-set calibration

Among the 93 Qwen disagreements, the judge accepts two and rejects 91. The 83 explicit wrong-objective-direction repairs are all rejected, identifying objective-direction restoration as a central semantic criterion. The positive controls receive ten accepts out of twelve; the missing-payload controls remain unsure. This audit provides blinded model-based evidence about the acceptance rules, alongside deterministic replay of the underlying verifier traces.

G.7 FRESH SAMPLED REPAIR-SEMANTIC AUDIT

We drew a simple random sample without replacement from the 360 Qwen RSS+normalizer hardening cases. The 117 cases represented in the prior acceptance audit were excluded, leaving 243 eligible cases; the frozen sample contains 80 cases (seed 20260925). Each blinded card includes the source-task requirements, original and perturbed instance, observed verifier feedback, and proposed repair. It omits benchmark scores, injected-error labels, reference repair targets, and prior audit labels. GPT-6 Astra, accessed through the AIHub API, returned a valid judgment for all 80 cards using the fixed three-way rubric (accept, reject, or uncertain).

The model returned 12 accepts, 37 rejects, and 31 uncertain judgments. Among the 49 decisive judgments, 42 agreed with strict RRR (85.7%). All 12 accepts were strict successes. Of the 37 rejections, 30 were strict failures and seven were strict successes. Uncertain judgments remain in the verdict counts and are excluded from decisive agreement. Strict RRR supplies the operational score, while GPT-6 Astra supplies blinded model-based semantic judgments. Because the audit uses one model, it does not estimate inter-model agreement.

Semantic-v2 therefore requires objective-direction repairs to restore the reference direction rather than merely mention the relevant concept. It rejects all 83 explicit wrong-direction cases and raises same-set agreement without changing the strict rule or strict results. The retained historical non-full-controller payloads do not support a uniform semantic-v2 rescore of the complete model fleet; original semantic-v1 tables therefore remain version-labeled. A fleet-wide semantic-v2 rescore and domain-expert assessment of construction oracles extend the current evaluation.