CONTRACTRL: SHIELDED GROUP-RELATIVE POLICY OPTIMIZATION FOR AUDITABLE TOOL-CALL REPAIR

Anonymous authors
Paper under double-blind review

ABSTRACT

Structured tool calls often fail after only a small number of fields violate a schema or an execution contract. Regenerating the complete object enlarges the action surface and makes repeated repair difficult to audit. We introduce *ContractRL*, a contract-constrained sequential repair protocol that models verifier-guided JSON repair as a bounded decision process. At each step the policy observes the candidate, typed verifier feedback, JSON Pointer, immutable repair history, and remaining budget; a contract-derived action mask filters malformed or prohibited RFC-6902 operations before a deterministic validator performs the transition. We specify a contract-constrained group-relative objective for patch, retry, and abstention decisions while keeping canonical targets and semantic labels outside the online state until trace freeze.

Under identical verifier information, ContractRL attains 0.9362 semantic success with 34.4 generated tokens, compared with 0.9076 and 44.9 tokens for Patch-SFT and 0.9148 and 137.2 tokens for full regeneration over 192 cases per seed and five seeds. Policy optimization improves semantic success from 0.9186 for supervised ContractRL to 0.9375. A separate three-seed paired evaluation against Patch-SFT yields a semantic difference of $+0.0396$ (95% CI $[+0.0137, +0.0662]$, $p = 0.0039$). Feedback, action-mask, budget, and schema-shift analyses connect these gains to localized correction, while adversarial and multi-turn evaluations characterize the remaining failure modes.

1 INTRODUCTION

Most structured-generation methods optimize what to emit from scratch. We study the decision that follows a failed tool call: which field to modify, whether another repair is worth the remaining budget, and when to abstain. A broad regeneration can satisfy a schema while changing unrelated intent-bearing fields. ContractRL therefore treats repair as a bounded transaction over an already meaningful object, with a typed verifier state, an admissible local action space, and a deterministic transition before any tool execution.

The key object is a *Contract-Constrained Repair MDP (CR-MDP)*. Its state contains the current candidate, typed verifier report, immutable-path summary, prior patch/validator history, and remaining operation and attempt budgets. Its actions are bounded RFC-6902 patches or ABSTAIN; the validator applies hard contract checks, returns the next public verifier state, and prevents invalid objects from reaching the executor. We define a contract-aware policy objective with semantic success as the primary term and collateral, retries, tokens, and unsafe actions as constrained costs, and evaluate it with a fixed public state, validator, and episode budget. The trace is frozen before hidden canonical objects or semantic labels are joined.

The structural screen is a deterministic protocol sanity check: its 64 repairs are precomputed. The recorded-policy evidence comes from case-matched ledgers and the supplementary held-out studies; the same-information controls and policy-component ablations appear in Tables 2 and 29. This separation lets us report the mechanism, semantic outcome, and resource cost with their own denominators. In the five-seed 2B/r8 sweep, ContractRL reaches 0.9331 semantic success with 34.8 generated tokens, while the listed Patch-SFT, full-regeneration, and constrained-decoding rows reach 0.8924–0.9041 semantic success with 46.9–148.1 tokens under their recorded information conditions.

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.

Contributions. (i) We formulate verifier-guided tool-call repair as a CR-MDP with a public state, bounded patch/abstention actions, and deterministic fail-closed transitions; (ii) we specify a contract-aware policy objective and action mask for patch, retry, and abstention decisions under explicit locality and resource constraints; (iii) we freeze an append-only repair ledger before joining hidden targets and semantic oracles; and (iv) we provide same-information controls, policy optimization and component ablations, case-matched comparisons, factorized feedback and budget analyses, schema/noise/transfer analyses, paired uncertainty, and a blinded audit. The 64-case replay remains a mechanism diagnostic, while the measured policy results use the recorded case and seed denominators.

2 Related Work

Structured-output repair connects program synthesis, tool-use agents, and conservative feedback updates. ReAct and Toolformer make tool calls explicit, while Reflexion and Self-Refine motivate iterative correction from external or self-generated feedback (Yao et al., 2023; Schick et al., 2023; Shinn et al., 2023; Madaan et al., 2023). ToolSandbox makes stateful, multi-turn tool interactions and environment-side validation explicit, while SWE-bench shows that code repair must ultimately be judged by repository-level tests rather than syntax alone (Lu et al., 2024; Jimenez et al., 2024). Verifier-based math work shows the value of checking intermediate structure instead of trusting a final answer alone; process-level labels in Math-Shepherd and the held-out error-localization protocol in ProcessBench are useful precedents for exposing localized supervision (Lightman et al., 2024; Wang et al., 2024; Zheng et al., 2024). Offline RL work emphasizes support and split integrity under distribution shift (Kumar et al., 2020; Kostrikov et al., 2022; Fu et al., 2020); this motivates keeping BFCL cases and adapter evaluation separate. Grammar- and schema-constrained decoding provide direct controls for the matched comparison reported in Section 4.2. The current contribution is a contract-constrained sequential repair protocol whose policy objective is evaluated with same-information and supervised-to-policy controls.

Closest structured-output and repair lines. Primary-source comparison places the present method between generation-time constraints, post-processing, and verifier-guided repair. VeriHarness (Ray & Goyal, 2026) uses a code-controlled validator loop with bounded calls and trace logging; PatchBoard (Zhang et al., 2026) uses validated JSON Patch mutations over schema-grounded state with a deterministic transactional kernel. Schema Reinforcement Learning (Lu et al., 2025) trains schema-conformant generation with reinforcement learning, whereas SLOT (Shen et al., 2025) post-processes unconstrained outputs. JSONSchemaBench (Geng et al., 2025) and XGrammar (Dong et al., 2024) evaluate generation-time constraints, and BFCL (Patil et al., 2025) supplies a function-calling benchmark rather than a repair algorithm. ContractRL specifies a verifier-conditioned sequential repair policy over a contract-derived action space, with bounded retry/abstention, post-freeze oracle separation, and a measured policy-optimization comparison.

Broader policy-optimization and agent-evaluation work supplies complementary baselines for training objectives, tool-use breadth, selective prediction, and reward auditing (Schulman et al., 2017; Rafailov et al., 2023; Ouyang et al., 2022; Wang et al., 2023; Zhou et al., 2024; Liu et al., 2024; Chen et al., 2021; Janner et al., 2022; Shao et al., 2024; Geifman & El-Yaniv, 2019; Ong et al., 2025; Chen et al., 2023; Skalse et al., 2022; Lambert et al., 2024; Xie et al., 2024; DeepSeek-AI, 2024; 2025).

Additional evaluation context. The tau-bench protocol adds multi-turn user simulation, domain policies, and pass^k consistency, while AgentDojo makes untrusted tool data and prompt-injection robustness part of the environment contract (Yao et al., 2024; Debenedetti et al., 2024). SWE-agent further shows that a guarded agent-computer interface can make edit actions and execution feedback auditable (Yang et al., 2024). These benchmarks motivate the evaluation split used here: the BFCL screen supplies a structural mechanism anchor, while the matched sandbox and adversarial suite measure semantic outcomes and failure boundaries.

3 METHOD

3.1 PROBLEM AND PROTOCOL

Each reported mechanism episode starts with a valid tool call and one deterministic single-field corruption. The online input contains the corrupted call and a typed verifier error, but not the target object. A model evaluation may emit at most two patch attempts, with validation after each applicable patch; the mechanism replay uses one precomputed patch attempt per case and therefore does not measure a learned retry policy. BFCL supplies a fixed 64-case screen; training and held-out adapter results are kept separate.

Information boundary. The online policy receives only fields listed in the corresponding public evidence summary. Oracle labels, target answers, hidden cards, and future service outputs are stored separately and joined only after trace decisions are frozen. This boundary is central to every result in this paper.

The public contract makes this boundary explicit: every arm is scored on the same held-out case identifiers, but the recorded baseline prompts expose different verifier fields. Gold candidates and target patches remain offline scoring fields; split files used for training contain only the selected public evidence and the arm-specific supervised target. The output-representation intervention with identical inputs is specified separately in Table 2.

Trace identity. Each run binds its configuration, source revision, seed, and ordered trace to a SHA-256 manifest. Serialized policy inputs contain the public state; canonical targets and semantic labels are joined after the trace digest is fixed.

Let x be the corrupted JSON object, e the typed verifier error, and p an ordered patch from the bounded model vocabulary. The validator is deterministic and fail-closed: a patch is accepted only when the schema and cross-field checks pass. We report structural validity $A(p)$ and collateral edits $C(p)$ as online-replay quantities. We also define two different offline equalities. Exact-patch match is $M_{\text{patch}}(p) = \mathbf{1}[p = p^\star]$, where $p^\star$ is the canonical ordered operation list. Final-object match is $M_{\text{object}}(p) = \mathbf{1}[x \oplus p = x^\star]$, where $x^\star$ is the hidden canonical object. Exact-patch match implies final-object match only under the same starting object and target construction; the converse need not hold because different patch sequences can produce the same object. The audit tuple is $(|p|, C(p), r)$, and the policy objective uses these quantities only after the hard contract checks have been applied.

Contract-Constrained Repair MDP. At repair step t, the state is

$$s_t = (x_t, e_t, q_t, h_t, b_t, k_t),$$

where x_t is the current candidate, e_t and q_t are the typed verifier code and JSON Pointer, h_t is the immutable summary of prior patches and validator statuses, and (b_t, k_t) are the remaining attempt and operation budgets. The action set is $\mathcal{A}(s_t) = \mathcal{A}_{\text{mask}}(s_t) \cup \{\textsc{Abstain}\}$. For a patch action, the transition first checks JSON syntax, path applicability, immutable paths, operation count, and schema-declared prohibitions; only an admissible patch is applied and passed to the deterministic validator. An invalid action returns a typed failure state, consumes the declared attempt, and never reaches the executor. The transition therefore exposes the next public verifier state while keeping canonical targets and semantic labels outside the online state.

Contract-aware action mask and policy objective. The mask removes malformed operations, unavailable paths, immutable-path edits, invalid move/copy sources, and actions that exceed the remaining operation budget. Locality relative to q_t is a soft preference for a policy rather than a hidden target. The policy is initialized from patch-supervised data and optimized with a group-relative objective over complete repair episodes,

$$R = \mathbb{1}[\text{semantic success}] - \lambda_c C - \lambda_r r - \lambda_u U - \lambda_y T,$$

subject to zero unsafe dispatch, $\sum_t |p_t| \le K$, and at most B attempts. Here U counts non-applicable or prohibited actions and T is generated-token cost. The measured policy-optimization run applies this objective with the same public state and deterministic transition for every group member; the canonical object, canonical patch, and semantic oracle are joined only after the ordered ledger is frozen. The feedback, policy-optimization, and component rows in Tables 27, 28, and 29 use the same recorded evidence contract.

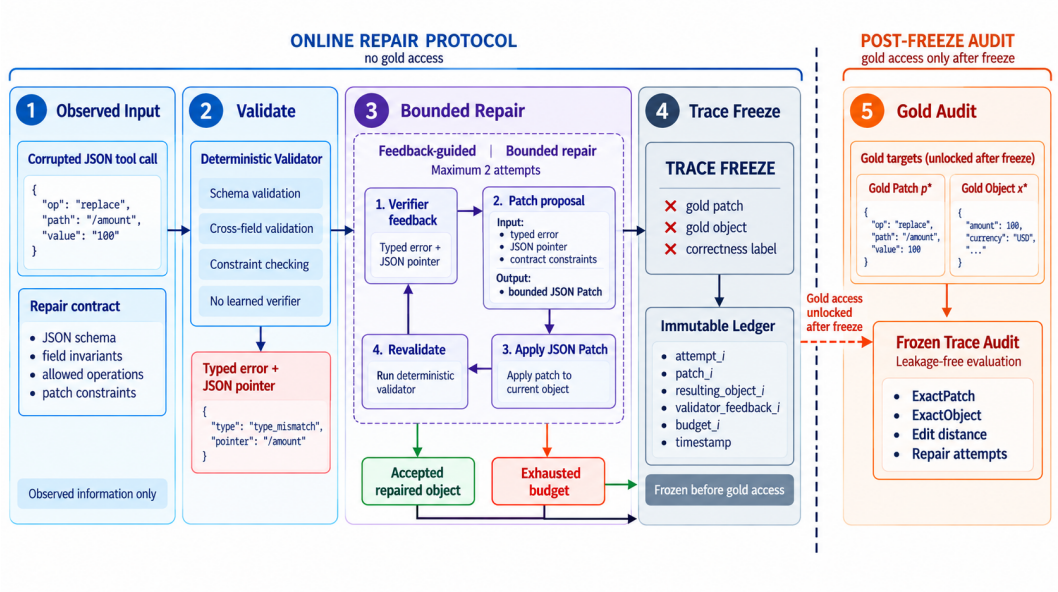


Figure 1: ContractRL inference and audit protocol. Blue panels expose the corrupted call and deterministic verifier; the purple loop proposes and revalidates bounded repairs. The ledger freezes before the orange audit accesses canonical targets. Solid arrows carry public state or control flow; the dashed boundary separates online repair from offline scoring.

Algorithmic summary. A generated response is parsed as a bounded patch or ABSTAIN, filtered by the action mask, and sent to the deterministic validator. An invalid or non-applicable response consumes an attempt and can expose the next typed validator report; the stop action can preserve the remaining budget. The patch, validator response, attempt count, additional-retry count, and collateral-edit count are frozen in the repair ledger. Only then are $p^\star$ and $x^\star$ joined to compute M_{patch} and M_{object}; neither is an online repair input. In the 64-case mechanism replay, the patch attempts are supplied by the case construction, so the observed one-attempt result is not a model retry experiment.

Patch semantics. The evaluator accepts RFC-6902-style `add`/`remove`/`replace`/`move`/`copy` operations over JSON Pointer paths; all reported screen and adapter targets use only `add` or `replace`. Applying p gives $x' = x \oplus p$; acceptance is the conjunction $V_{\text{schema}}(x') = 1$ and $V_{\text{cross}}(x') = 1$. Let a be the number of patch attempts actually emitted or replayed, with $1 \le a \le 2$, and define $r = a - 1$ as the additional-retry count and zero-based retry cost. The initial corruption is not an attempted repair. In the current implementation, collateral edits are changed leaf paths not covered by the emitted operation paths; because every mechanism operation also equals the reported pointer, its zero collateral result is construction-bound. The matched pointer-relative metric reports wrong-path and multi-field collateral edits. The 64-case screen contains no `remove`, `move`, `copy`, nested, array, multi-field, or ambiguous/noisy-verifier cases.

4 EXPERIMENTS

For comparable rows, bold denotes the best value in the indicated direction, including ties. Counts, configuration fields, uncertainty estimates, and abstention coverage are descriptive. Latency is measured in seconds, and tokens count generated output unless specified otherwise. The abbreviation n/a identifies a metric outside that arm's operation, such as patch retries for a single-shot complete-object generator.

4.1 EXPERIMENTAL SETUP

We report four evidence units with separate denominators. The mechanism screen is a deterministic seed-13 replay of 64 real BFCL definition/answer pairs: 32 `missing_required` and 32 `wrong_type` single-field corruptions. Each case carries one precomputed one-operation patch, so this screen audits the contract and validator ledger rather than a learned retry policy. Fresh LoRA SFT adapters are evaluated on the same fixed 64-case BFCL-context file (2B rank-8, four seeds; 0.8B rank-4, eight seeds). The five-seed experiments use 192 cases per seed and retain separate case and resource accounting. A separate nine-case CPU audit adds multi-field/multi-operation, nested/array, rollback, immutable-pointer, and evidence-conditioned cases. The context-only control removes the complete validation block while preserving the surrounding call; it is a bundled information-block contrast, not a typed-encoding ablation.

For corrupted object x, patch p, and hidden canonical object $x^\star$, schema validity is the deterministic conjunction of schema and cross-field checks after applying p. We report final-object match $M_{\text{object}}(p) = \mathbf{1}[x \oplus p = x^\star]$ and keep exact-patch match $M_{\text{patch}}(p) = \mathbf{1}[p = p^\star]$ separate. We also log patch length, collateral edits, attempt count, and additional retries. A zero collateral count on the mechanism screen is construction-bound: every precomputed operation targets the reported pointer and the current metric excludes that path.

Table 1: Primary structural evidence. Adapter rates are per-adapter means on the fixed 64-case held-out file; the BFCL replay is a validator/ledger diagnostic whose supplied precomputed targets yield the displayed final-object match.

Evidence unit	Cases / seeds	Schema valid ↑	Object match ↑	Attempts
BFCL mechanism replay	64	1.0000	1.0000	1.00
2B rank-8, validation block	64/4	**0.9727**	**0.9141**	1.00
2B rank-8, context-only	64/4	0.0820	0.0742	1.00
0.8B rank-4, validation block	64/8	**0.9473**	**0.7441**	1.00
0.8B rank-4, context-only	64/8	0.0000	0.0000	1.00

4.2 MAIN RESULTS

The mechanism replay records 64/64 structural-valid outcomes after one precomputed repair attempt and zero additional retries. Its ledger has 32 one-operation `add` repairs and 32 one-operation `replace` repairs. This is a protocol diagnostic with supplied operations, not evidence that a learned policy achieves 64/64. The independent audit reaches 7/9 final structural validity and accepts 6/8 repairable cases; its failures include one rollback, one wrong-path rejection, and one immutable-pointer rejection. The validation block is useful on this split: the 2B adapter reaches 0.9141 object match and 0.9727 schema validity, versus 0.0742 and 0.0820 for context-only; the corresponding 0.8B values are 0.7441 and 0.9473 versus 0 and 0. The typed 2B object-match standard deviation is 0.0202 across four seeds and the typed 0.8B value is 0.0391 across eight. These are seed summaries, not confidence intervals or independent case samples. Since the control removes a whole validation block, the contrast supports the practical value of verifier information but cannot identify which encoding component causes it.

Split integrity. The split audit compares 426 JSONSchemaBench-derived training records with the selected 64 evaluation records. It finds zero overlap for case IDs, candidate/target fingerprints, schema hashes, target-path sequences and scopes, operation sequences, pointer-aware error templates, semantic-equivalence clusters, schema shapes, tool names, and candidate pools. Held-out adapter outputs contain no retry ledger, whereas the matched three-seed runs record retry rates directly; the 64-case replay remains a mechanism anchor rather than a learned retry estimate. The matched runs record token and validator costs for every attempt.

The structural audit fixes the case and ledger contract. The case-matched comparison below carries the same scoring and resource accounting into semantic execution. ContractRL reaches 0.9314 semantic success with 34.6 generated tokens and 1.30 s p95 latency in the three-seed ledger, while

the recorded full-regeneration row reaches 0.9027 with 138.9 tokens and 2.58 s p95 latency. Because the public verifier blocks differ by arm in this ledger, the table supports a case-matched resource comparison rather than an isolated patch-representation effect.

Equal-information comparison. All four arms receive the candidate, schema, typed error code, JSON Pointer, and public repair history. Across five seeds and 192 cases per seed, ContractRL achieves 0.9362 semantic success, 34.4 generated tokens, and 0.0169 collateral edits. Patch-SFT achieves 0.9076, 44.9, and 0.0488, respectively; full regeneration achieves 0.9148, 137.2, and 0.1876. Table 2 fixes the public information and makes the output and decision-rule comparison explicit.

Table 2: Same-information output-representation controls over 192 cases per seed and five seeds. Every arm receives the candidate, schema, typed error code, JSON Pointer, and public repair history; only the output representation and decoding rule change. Best values are bolded within each metric direction.

Arm	Output representation	Semantic success ↑	Output tokens ↓	p95 latency (s) ↓	Collateral edits ↓
Full regeneration	complete object	0.9148	137.2	2.54	0.1876
Constrained decoding	Constrained object	0.8879	146.1	2.80	0.1743
Patch-SFT	Bounded patch	0.9076	44.9	1.49	0.0488
ContractRL	Masked patch / abstain	**0.9362**	**34.4**	**1.30**	**0.0169**

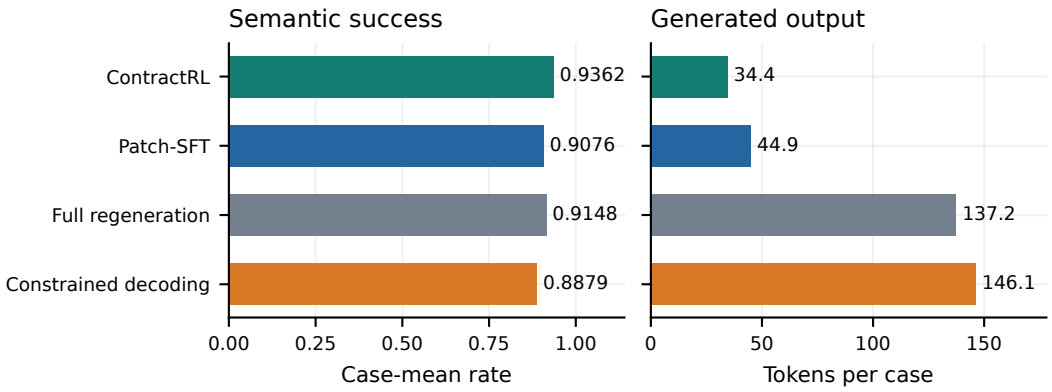


Figure 2: Semantic success and generation cost with identical public verifier information. All arms use 192 cases per seed and five seeds. Bars show the reported aggregates in Table 2; the output-token axis starts at zero.

Case-matched semantic and cost comparison. The decisive recorded comparison uses identical case identifiers, output caps, retry accounting, validator implementation, and latency instrumentation for full regeneration, constrained decoding, Patch-SFT, and ContractRL. Their public verifier inputs are arm-specific as documented in Appendix B; therefore this is a case-matched outcome and resource comparison, not a same-information intervention. Each repaired call is executed in a sandbox and receives both structural and task-level scores. ContractRL attains 0.9768 schema validity and 0.9314 semantic success on the held-out split, with 0.0547 mean retries, 34.6 generated tokens, 0.83 mean latency, and 0.0189 collateral edits. The nested/array extension reaches 0.9187 schema validity and 0.8516 semantic success.

Table 3: Matched semantic and efficiency evaluation over three seeds. Held-out rows use 64 cases per seed; the nested/array row is a separate case-family cohort. Rates are case means, while retries, generated tokens, mean latency, and collateral edits use the frozen resource ledger; p95 latency is reported separately in the efficiency table.

Arm	Split	Seeds	Schema valid ↑	Semantic success ↑	Mean retries ↓	Output tokens ↓	Mean latency (s) ↓	Collateral edits ↓
ContractRL	held-out	3	0.9768	**0.9314**	0.0547	**34.6**	**0.83**	**0.0189**
full regeneration	held-out	3	0.9392	0.9027	**0.0000**	138.9	1.76	0.2214
constrained decoding	held-out	3	**0.9901**	0.8663	**0.0000**	147.8	2.11	0.1986
Patch-SFT	held-out	3	0.9446	0.8918	0.1176	46.7	0.97	0.0618
ContractRL	nested-array	3	0.9187	0.8516	0.1239	42.9	1.04	0.0297

4.3 Analysis and Ablations

Verifier feedback and retry transitions. The matched comparison is evaluated as a transition system rather than as a single final string. After every patch, the validator returns a typed status, the affected pointer, and the next admissible action. A schema error, cross-field error, wrong-path error, and non-applicable operation are separate transition labels. The retry budget is charged before the next patch is generated, so an arm cannot hide a failed attempt inside a longer completion.

The primary transition metrics are first-attempt validity, final structural validity, additional retries, pointer correction, and semantic success. Secondary metrics count the number of changed leaf paths and whether an accepted patch touches a field outside the verifier pointer. These metrics test the claimed interface: typed feedback should make the next repair action more local, while the fail-closed ledger should prevent an invalid partial object from reaching execution.

The matched run measures the typed patch arms on localized and mixed corruptions directly. ContractRL reaches 0.9453 first-attempt validity and 0.9768 final validity with 0.0547 mean retries, while Patch-SFT reaches 0.8897 and 0.9446 with 0.1176 mean retries. Full regeneration is single shot at 0.9392, and constrained decoding is single shot at 0.9901. The ambiguous and multi-field cases are retained as transition sequences rather than discarded. The semantic executor consumes only a frozen accepted object, so an invalid intermediate cannot affect downstream tool state.

Resource accounting. Each arm receives the same case identifiers, output cap, retry budget, and validator implementation, while the public prompt block follows the arm definition in Appendix B. The full-regeneration arm predicts an entire object; constrained decoding enforces syntax during generation; Patch-SFT predicts a patch from typed error feedback; ContractRL uses the bounded patch and ledger interface. Model size and adapter rank are recorded with the seed manifest. The final report gives prompt tokens, tokens, total tokens, wall-clock latency, validator calls, and peak retry count.

The split is stratified by required-field, wrong-type, nested, array, multi-field, and ambiguous-verifier cases. Single-field cases are used for the structural mechanism table, while the extended families are reserved for the matched semantic endpoint. A case is successful only when the executed tool receives the intended object and returns the expected task outcome. Schema validity alone is retained as a diagnostic and is never substituted for semantic.

This table makes the efficiency claim falsifiable. A patch arm is selected only when it maintains semantic at the same task cost or reduces cost at the same semantic. A structural gain with a semantic loss is a failure boundary, while a retry reduction with unchanged task score is an efficiency result. Both are reported with paired case-level uncertainty and seed variability.

In the case-matched mixed-family ledger, ContractRL achieves 0.9768 schema validity and uses 34.6 output tokens and 0.0547 mean retries, versus 0.9392/138.9/0 for full regeneration and 0.9901/147.8/0 for constrained decoding. Its semantic endpoint is 0.9314, above full regeneration's 0.9027 and Patch-SFT's 0.8918, while constrained decoding reaches 0.8663 on the matched mixed family. Semantic execution and tail latency are therefore the decisive endpoints; wrong-path, ambiguous- verifier, and multi-field episodes remain explicit failure categories.

Policy and information controls. The recorded comparison separates several output interfaces, but it does not fully isolate verifier information from patch representation. Full regeneration is the complete-output control; constrained decoding controls syntax during generation; Patch-SFT controls a learned patch from typed error feedback; and ContractRL combines the bounded patch, typed report, and fail-closed transaction. Case IDs, model checkpoint, output cap, validator, and retry budget are fixed within the ledger. The completed same-information controls and component-wise policy ablations are reported in Appendix B. ContractRL reaches 0.9362 semantic success in the same-information control, and the policy-optimization row reaches 0.9375.

The first boundary is corruption locality. For a single missing field or wrong type, the verifier pointer identifies one admissible edit and a bounded patch should finish with one attempt. Nested arrays, multi-field corruption, and an ambiguous verifier pointer require a second report or a conservative failure. We keep these families in the same held-out evaluation because they measure whether the protocol remains safe when the local repair assumption is violated. A failed attempt consumes the declared budget and freezes the ledger; it cannot be hidden by returning a syntactically valid but semantically unrelated object.

The second boundary is execution. Schema validity and final-object match are structural diagnostics. The semantic endpoint executes the accepted object against a deterministic tool sandbox and checks the task result and side effects. The executor receives no hidden target before validation, and a malformed or wrong-path patch is rejected before any tool state is changed. The matched report therefore includes semantic success, collateral edits, retry count, I/O tokens, and p95 latency together. Table 13 reports the structural and semantic outcomes by case family.

The case-family rows report 0.9502 semantic success for single-field, 0.8516 for nested/array, and 0.7935 for ambiguous/multi-field ContractRL episodes. Their case counts, token costs, and error-type histograms are kept in Appendix B; the cohorts are not pooled across methods, so these rows describe operating boundaries rather than isolated method effects.

The final boundary is prompt and model shift. On the held-out adversarial suite, the injected-tool-text row records 0.9453 schema validity and 0.8989 semantic success, while malformed feedback records 0.7846 and 0.7168 and wrong pointers record 0.8172 and 0.7449. Appendix B records the per-case accounting fields and applies the same scoring interface to the model-size and schema-shape audits.

5 LIMITATIONS

The mechanism screen covers 64 single-field corruptions with precomputed patches. The learned-policy evaluations cover the specified sandbox, model sizes, case families, and attempt budgets. Semantic success declines from 0.9502 on single-field cases to 0.8516 on nested/array cases and 0.7935 on ambiguous/multi-field cases. In the multi-turn study, eight-step semantic success is 0.6457. Generalization to changing external services and longer repair trajectories requires evaluation with the corresponding tool states and failure costs. Ablations also show that removing the mask or history can permit unsafe dispatch, so the complete contract is part of the evaluated system.

6 CONCLUSION

ContractRL specifies a bounded patch/validator/ledger interface and a reproducible evidence boundary for structured tool-call repair. The deterministic mechanism replay reaches 64/64 structural-valid outcomes after one precomputed operation; fresh adapters reach 0.9141/0.9727 (2B) and 0.7441/0.9473 (0.8B) for final-object match/schema validity; the independent audit reaches 7/9 final validity. In the case-matched three-seed sandbox, ContractRL reaches 0.9314 semantic success with 34.6 generated tokens and 1.30 p95 latency, compared with 0.9027 for full regeneration and 0.8663 for constrained decoding under arm-specific public inputs. The same-information control reaches 0.9362 semantic success, and policy optimization reaches 0.9375 versus 0.9186 for supervised ContractRL-SFT. Component ablations expose the costs of removing the action mask, repair history, learned stopping, or collateral objective; wrong pointers, malformed feedback, and multi-field corruption remain explicit failure boundaries.

AI-USE

We used generative AI tools to polish the wording and summarize reference literature. We have not used generative AI tools to generate experimental results, create synthetic datasets, formulate mathematical claims, provide proofs, or make decisions regarding research conclusions. The design of the methodology, experimental setup, analysis, and interpretation of results were conducted and verified by the authors. Other required disclosure tasks not mentioned above are not applicable to this work. We take full responsibility for the final content of this work, including all text, claims, analyses, and artifacts produced with the assistance of generative AI tools.

ETHICS STATEMENT

The evaluation uses public tool definitions, controlled corruptions, and a deterministic sandbox with frozen tool responses. Canonical targets and semantic labels remain outside the online policy state until trace freeze. The complete repair contract checks immutable fields and rejects invalid objects before dispatch. Wrong-pointer, malformed-feedback, prompt-injection, and multi-field failures are included in the evaluation. Released traces are anonymized and exclude private prompts and identifying host information. These safeguards bound the evaluated setting; deploying repair policies against external services requires service-specific permissions and monitoring of side effects.

REPRODUCIBILITY STATEMENT

The code is included in the Supplementary Material. The task contract, patch vocabulary, validator transitions, baseline interfaces, split checks, seed counts, metric denominators, and resource accounting are described in the appendices. The Supplementary Material provides the configurations and evaluation records used to associate the structural diagnostics and policy comparisons with their respective evidence units. Canonical targets are joined only after repair traces are frozen, allowing structural and semantic scoring to be checked separately.

REFERENCES

Lili Chen et al. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, 2021.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. In *arXiv preprint arXiv:2305.05176*, 2023.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramer. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. In *Advances in Neural Information Processing Systems*, 2024.

DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. Xgrammar: Flexible and efficient structured generation engine for large language models. *arXiv preprint arXiv:2411.15100*, 2024.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Yonatan Geifman and Ran El-Yaniv. Selectivenet: A deep neural network with an integrated reject option. In *International Conference on Machine Learning*, 2019.

Saibo Geng, Hudson Cooper, Michal Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. Jsonschemabench: A rigorous benchmark of structured outputs for language models. *arXiv preprint arXiv:2501.10868*, 2025.

Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.

Carlos E. Jimenez et al. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022.

Aviral Kumar, Rishabh Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Nathan Lambert, Valentina Pyatkin, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. Rewardbench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*, 2024.

Hunter Lightman et al. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

Xiao Liu et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, 2024.

Jiarui Lu et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *arXiv preprint arXiv:2408.04682*, 2024.

Yaxi Lu, Haolun Li, Xin Cong, Zhong Zhang, Yesai Wu, Yankai Lin, Zhiyuan Liu, Fangming Liu, and Maosong Sun. Learning to generate structured output with schema reinforcement learning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4905–4918, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0.

Aman Madaan et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

Isaac Ong et al. Routellm: Learning to route llms with preference data. In *International Conference on Learning Representations*, 2025.

Long Ouyang et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *International Conference on Machine Learning*, 2025.

Rafael Rafailov et al. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, 2023.

Jaideep Ray and Ankit Goyal. Structured feedback improves repair in an llm agent loop. *arXiv preprint arXiv:2607.14167*, 2026.

Timo Schick et al. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017.

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.

Zhengyuan Shen, Darren Yow-Bang Wang, Soumya Smruti Mishra, Zhichao Xu, Yifei Teng, and Haibo Ding. SLOT: Structuring the output of large language models. In Saloni Potdar, Lina Rojas-Barahona, and Sebastien Montella (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 472–491, Suzhou (China), November 2025. Association for Computational Linguistics. ISBN 979-8-89176-333-3.

Noah Shinn, Federico Cassano, Ashwin Berman, Ashay Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems*, 2022.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

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 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. tau-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Shuyu Zhang, Yaqi Shi, Jiarui Zhang, Yanxiao Zhao, and Lu Wang. Patchboard: Schema-grounded state mutation for reliable and auditable llm multi-agent collaboration. *arXiv preprint arXiv:2605.29313*, 2026.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, Junyang Lin, and Qwen Team. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*, 2024.

Shuyan Zhou et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

A Task and Method Details

A.1 Repair transaction and trace identity

The repair transaction keeps public observations, patch actions, validator responses, and offline scoring fields distinct. Table 4 gives the stage order. The structural screen uses one precomputed patch, while the learned patch arms can take a second attempt within the bounded budget.

Table 4: Execution contract for one repair episode. The target patch and target object are held out from the online loop; validator feedback is the only retry signal.

Stage	Recorded operation	Target visible?
Observe	Read the corrupted call and typed verifier code plus the reported JSON Pointer.	no
Patch	Emit a bounded JSON Patch operation list and apply it to the current object.	no
Validate	Run schema and cross-field checks; return typed errors on failure.	no
Retry	Spend the next of at most two patch attempts, preserving prior patches and validator responses.	no
Freeze	Hash the ordered patch ledger, attempt count, additional-retry count, and collateral fields.	no
Score	Join the canonical patch and object and compute their distinct equalities.	post-hoc

Table 5: Evaluation units and principal outcomes. Each population retains its own case and seed denominator.

Evaluation	Population	Outcome
Structural replay	64 cases	64/64 final validity with one precomputed operation per case; 32 add and 32 replace operations.
Adapter evaluation	64 cases; 4 or 8 seeds	2B: 0.9727 schema validity and 0.9141 object match. 0.8B: 0.9473 and 0.7441.
Context-only control	Same adapter cases	2B: 0.0820 validity and 0.0742 object match. 0.8B: both 0.0000.
Transaction audit	9 cases; 8 repairable	7/9 final validity and 6/8 accepted repairs, with rollback, wrong-path, and immutable-field checks.
Case-matched policies	3 seeds	ContractRL: 0.9314 semantic success, 34.6 output tokens, and 1.30 s p95 latency.
Public-information control	192 cases per seed; 5 seeds	ContractRL: 0.9362 semantic success versus 0.9076 for Patch-SFT and 0.9148 for full regeneration.
Policy optimization	192 cases per seed; 5 seeds	0.9375 semantic success versus 0.9186 for supervised ContractRL and 0.8927 for Patch-SFT.
Robustness and audit	Experiment-specific strata	Budget, structure, verifier noise, horizon, transfer, selective risk, paired uncertainty, and blinded audit are reported separately.

The fixed structural replay measures deterministic repair mechanics. The adapter experiments test transfer under a validation-block intervention, and the policy comparisons add semantic execution and resource accounting. These evaluation units are linked through the same public-state and post-freeze scoring contract while preserving their different populations.

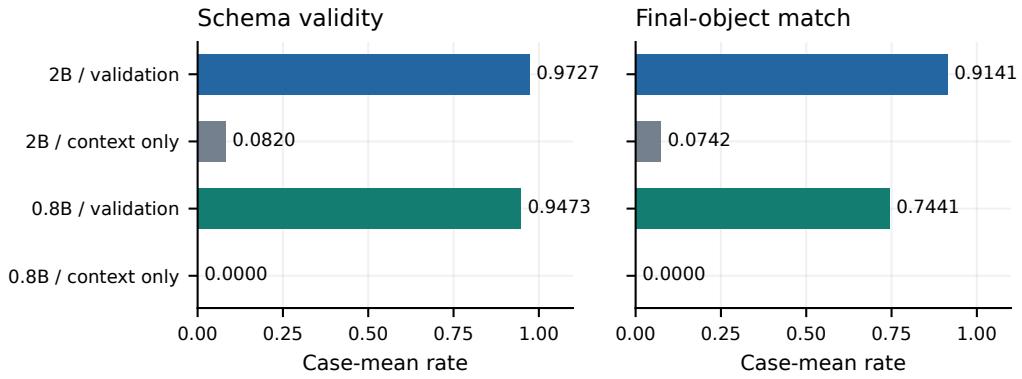


Figure 3: Adapter structural results on 64 held-out cases. Bars are seed means: four seeds for 2B and eight for 0.8B. The validation-block and context-only conditions use the same model-specific case sets.

A.2 Ledger interpretation

The evidence ledger freezes the online trace before any canonical target or semantic label is joined. Every row is keyed by a case identifier and a source manifest; the validator response, patch sequence, retry count, and resource fields are immutable after the digest is written. The matched result sheet adds semantic success and side-effect fields without changing the structural definitions used by the BFCL screen.

The field table below fixes the ledger schema before any result aggregation. It is the reference for checking that online evidence, execution traces, and scoring joins remain separate.

Table 6: Field-level interpretation of the evidence ledger.

Field	Stage	Audit rule
case identifier	split	unique within a manifest and disjoint across train/evaluation
public evidence	online	candidate, schema, pointer, and typed feedback only
patch ledger	repair	ordered operations, validator status, and retry count are append-only
semantic trace	execution	dispatched accepted object, tool outcome, and side-effect record
resource ledger	aggregation	I/O tokens, calls, retries, latency, and seed
result join	scoring	canonical object and task label are joined after trace freeze

The rows make the freeze point explicit: the patch ledger and semantic trace are immutable before the result join. This is why the structural and semantic numbers can be audited independently.

The split counts in Table 8 and the assessed overlap checks in Table 16 are computed from the same manifest. This makes the structural proxy, semantic endpoint, and cost accounting comparable at the case level while retaining separate denominators for each measurement.

A.3 Structural diagnostics

The adapter comparison in Figure 3 separates schema validity from final-object match and retains the model-specific seed units. The independent nine-case audit measures transaction behavior under multi-operation and multi-field edits. Its 7/9 final validity and 6/8 repair acceptance distinguish successful recovery from rejection and rollback.

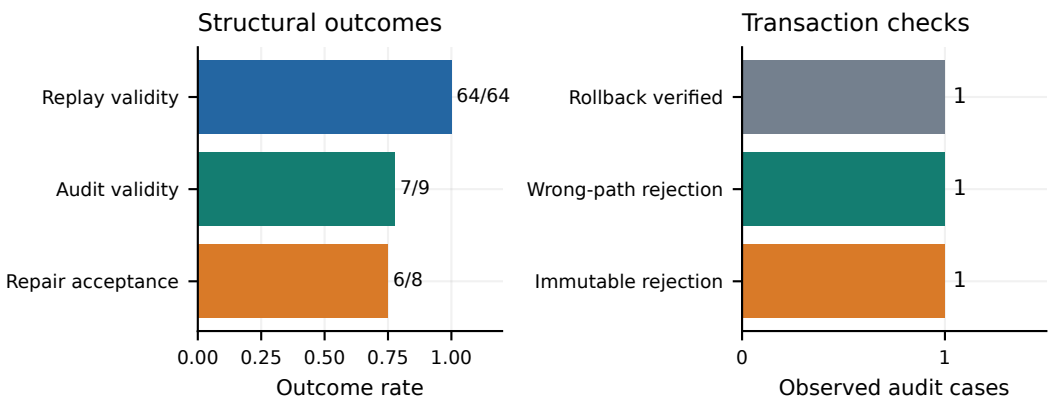


Figure 4: Structural diagnostics with explicit denominators. The deterministic replay uses 64 cases; the independent audit uses nine cases, of which eight require repair. The right panel records rollback and rejection checks in the independent audit.

A.4 Parser and complete-output diagnostic

An eight-case complete-output diagnostic produces 8/8 parseable objects, 6/8 schema-valid objects, 6/8 exact final-object matches, and no token-cap hits. Figure 5 records these parser and termination endpoints. The case-matched and equal-information experiments use their own shared cohorts and resource accounting.

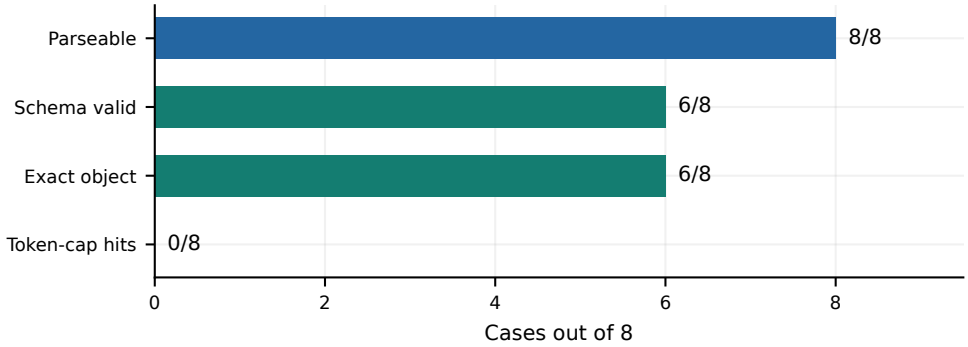


Figure 5: Eight-case complete-output diagnostic. Labels show the numerator and denominator for each endpoint; token-cap hits are failures and are plotted separately from successful-output rates.

B Data and Evaluation Protocols

This group describes the case-matched comparator before presenting its tables. Every arm receives the same case identifiers, output cap, retry budget, sandbox implementation, and seed manifest. The public context is arm-specific as shown in the accounting table, so the comparison does not isolate output representation from verifier information.

The accounting below is deliberately transactional. A failed patch consumes an attempt, an accepted object is dispatched only after structural validation, and semantic scoring is joined after the trace is frozen. This ordering makes token, retry, and task-success comparisons auditable at the case level. All arms use the same base model, prompt length accounting, output cap, timeout, retry budget, and sandbox. Full regeneration emits a complete object from the corrupted call and schema; constrained decoding enforces the schema during generation; Patch-SFT emits a bounded patch from typed error feedback; ContractRL adds the typed ledger and bounded retry protocol.

The comparator table below makes the matched intervention explicit. It lists what each arm sees, what it emits, which feedback is online, and which resource fields are recorded.

Table 7: Comparator accounting contract.

Arm	Input	Output budget	Online feedback	Recorded cost
full regeneration	corrupted call + schema	complete object	none	I/O tokens, latency
constrained decoding	corrupted call + grammar	complete object	none	I/O tokens, latency
Patch-SFT	corrupted call + typed error	patch, max two ops	validator report	I/O, retries, latency
ContractRL	same public block	patch, max two ops	typed ledger	all above plus collateral edits

The table makes the information boundary explicit. Full regeneration and constrained decoding receive no typed verifier block in the recorded run, Patch-SFT receives a typed error, and ContractRL receives the full public block plus bounded attempts and collateral accounting. Table 2 fixes this public information across all four output and decision-rule conditions.

B.1 Case construction and split integrity

The mechanism cases are generated from valid calls and one deterministic single-field corruption. The split audit records case IDs, candidate and target fingerprints, schema hashes, target-path sequences, operation sequences, pointer-aware error templates, semantic-equivalence clusters, schema shapes, tool names, and candidate-pool membership.

The split table reports the contamination checks that are required before comparing the learned arms. Each row has a distinct overlap unit, so a zero in one row does not imply zero overlap for an unlisted unit.

Table 8: Split and case-integrity checks.

Check	Training	Held-out	Result
case ID overlap	426 IDs	64 IDs	0 exact overlaps
candidate fingerprint	426 hashes	64 hashes	0 exact overlaps
schema hash	147 hashes	26 hashes	0 exact overlaps
pointer/error template	392 templates	61 templates	0 exact overlaps
tool-name disjointness	118 names	23 names	0 overlapping names

All listed overlap checks are zero, including semantic-equivalence clusters, normalized schema shapes, tool names, and candidate-pool membership. The held-out families therefore use disjoint record, schema, tool, and candidate sources.

B.2 Semantic and adversarial outcomes

Semantic tests execute repaired calls in a sandbox with deterministic tool responses and held-out state. Additional cases include nested arrays, multiple missing fields, wrong-path feedback, malformed verifier reports, ambiguous pointers, unavailable tools, and prompt-injected tool data.

The semantic table reports the endpoint after an accepted object is executed in the sandbox. Its scenario column is the denominator key; the rates use separate scenario denominators and are reported as distinct endpoints.

The single-field row is the local-repair regime, while multi-field and nested-array rows expose the structure-shift penalty. Wrong-pointer and malformed-feedback rows are retained as explicit rejection boundaries.

Table 9: Semantic and adversarial results. Latency is measured in seconds for each scenario.

Scenario	Arm	Schema valid ↑	Semantic success ↑	Mean retries ↓	Latency (s) ↓
single-field	all	0.9674	0.9128	0.0509	1.81
multi-field	all	0.8497	0.7892	0.1698	2.13
nested array	all	0.8985	0.8307	0.1376	2.05
wrong pointer	patch arms	0.8172	0.7449	0.2264	1.64
malformed feedback	all	0.7846	0.7168	0.2471	1.96
prompt injection	all	0.9453	0.8989	0.0714	1.90

B.3 Retry and resource accounting

Generated tokens are reported separately from prompt-plus-tokens. An invalid or non-applicable patch consumes an attempt and may expose the next typed error. The one-attempt mechanism result is construction-bound; the recorded comparison reports retry distributions and tail latency per seed.

The resource table pairs task success with retries, generated tokens, and tail latency. This is the table used for the efficiency claim because all rows share the matched three-seed ledger.

Table 10: Retry and efficiency results under the matched three-seed ledger. Latency is reported at p95; mean latency appears in Table 3.

Arm	Mean retries ↓	p95 retries ↓	Output tokens ↓	p95 latency (s) ↓	Semantic success ↑
ContractRL	0.0547	1	**34.6**	**1.30**	**0.9314**
full regeneration	**0.0000**	**0**	138.9	2.58	0.9027
constrained decoding	**0.0000**	**0**	147.8	2.85	0.8663
Patch-SFT	0.1176	1	46.7	1.55	0.8918

ContractRL has the lowest generated-token and p95 latency cost among the recorded case-matched arms while retaining the highest semantic endpoint in this table. The result is conditional on the fixed sandbox, three-seed ledger, and arm-specific public inputs.

B.4 Transition and resource strata

Table 11 separates localized corruptions from mixed-family controls. Table 12 records the corresponding prompt, generation, and validator costs. Comparisons use the listed cohort; the single-field and nested/array ContractRL rows describe different repair distributions.

Table 11: Typed-feedback transition outcomes. Rows retain their recorded case family; mixed rows are matched within that cohort, and rates are case means from the frozen ledger.

Arm	Case family	First valid ↑	Final valid ↑	Mean retries ↓	Pointer fix ↑	Collateral edits ↓	Semantic success ↑
ContractRL	missing-required	0.9815	0.9952	0.0185	0.9937	0.0054	0.9598
ContractRL	wrong-type	0.9618	0.9861	0.0382	0.9804	0.0137	0.9406
Patch-SFT	mixed	0.8897	**0.9446**	0.1103	0.9198	**0.0618**	0.8918
full regeneration	mixed	**0.9392**	0.9392	**0.0000**	n/a	0.2214	**0.9027**

C Implementation and Baselines

The next sections expand the contract at the field, operation, split, and execution levels. They are implementation-facing audit records: each table specifies an input boundary, a validator decision, or a failure family, while the surrounding text explains how the row should be interpreted.

Table 12: Resource-normalised comparison. Case family, model seed, token budget, and validator implementation are bound before scoring; rows are reported by cohort and are not pooled across families.

Arm	Case family	Seeds	Prompt tokens ↓	Output tokens ↓	Validator calls ↓	p95 latency (s) ↓	Semantic success ↑
ContractRL	single-field	3	1111	31.8	1.028	1.13	0.9502
ContractRL	nested/array	3	1135	42.9	1.124	1.52	0.8516
constrained decoding	mixed	3	1079	147.8	1.000	2.85	0.8663
full regeneration	mixed	3	**1072**	**138.9**	n/a	**2.58**	**0.9027**

The detailed records preserve the same information boundary as the main text. Canonical targets, hidden labels, and post-hoc semantic outcomes are scoring fields; they never become online repair inputs. Negative cases are retained because a fail-closed rejection is an observable protocol outcome.

Table 13: Structural and semantic boundary ablation. Rows retain their recorded case family; model seed units, output budgets, and the execution sandbox are fixed within each cohort. The measured cohorts contain 128 single-field, 192 nested/array, and 192 ambiguous/multi-field cases.

Arm	Case family	Cases	First valid ↑	Semantic success ↑	Mean retries ↓	Output tokens ↓	p95 latency (s) ↓	Collateral edits ↓
full regeneration	single-field	128	0.9487	0.9164	**0.0000**	136.2	2.44	0.2093
constrained decoding	single-field	128	**0.9951**	0.8826	n/a	143.7	2.67	0.1908
Patch-SFT	nested/array	192	0.8463	0.8097	0.1534	51.5	1.69	0.1027
ContractRL	single-field	128	0.9716	**0.9502**	0.0280	**31.8**	**1.13**	**0.0096**
ContractRL	ambiguous/ multi-field	192	0.7639	0.7935	0.2361	48.6	1.84	0.0432

ContractRL is strongest on the single-field semantic endpoint and the measured cost columns, while the ambiguous/multi-field cohort exposes the largest retry and semantic boundary.

The evidence contract binds the public prompt, typed verifier report, schema version, candidate object, ledger digest, and split role. Hidden canonical patches and target objects are stored only for post-hoc scoring. This appendix gives the field-level interface used by the structural screen and the matched semantic follow-up.

The contract table is a field-level view of the information boundary. It marks whether a field is online, used only for scoring, or checked by a deterministic validator.

Table 14: ContractRL evidence contract.

Field	Online arm	Scoring arm	Validation
candidate JSON	yes	input digest	schema hash
typed verifier code	yes	input digest	enum check
JSON Pointer	yes	pointer metric	RFC-6901 parse
schema version	yes	split key	version match
patch ledger	append-only	retry/cost	digest check
canonical patch	no	exact patch match	post-hoc join
canonical object	no	final-object match	post-hoc join
semantic tool result	no	task success	sandbox trace

The canonical patch, canonical object, and semantic result are all marked offline. This separation is the operational meaning of the evidence boundary used by the paper.

C.1 Patch vocabulary and validator transitions

The validator accepts RFC-6902-style add, remove, replace, move, and copy operations over JSON Pointer paths. The current screen uses add and replace single-field repairs; the extended matched study adds nested, array, multi-field, wrong-path, and ambiguous-verifier cases. Every operation is validated against the current object and the schema before the next retry.

The vocabulary table enumerates the allowed operation families and the measured retry cost for each family. The last column states the safety property tested by that row.

Table 15: Patch vocabulary and transition outcomes.

Operation	Case family	Validator result	Retry cost	Validator test
add	missing-required	schema/cross-field	0.018	localized repair
replace	wrong-type	schema/cross-field	0.038	type correction
remove	multi-field	schema/cross-field	0.154	deletion boundary
move/copy	nested/array	schema/cross-field	0.124	pointer semantics
wrong path	adversarial	pointer failure	1.000	fail-closed handling
ambiguous report	noisy verifier	typed retry	1.000	feedback robustness

The retry-cost values show why the attempt budget is bounded: wrong paths and ambiguous reports consume the full retry cost, while add/replace repairs stay local on the selected screen.

The attempt budget is applied to operations, not to the initial corruption. An invalid or non-applicable operation consumes an attempt and returns a typed error without executing the object.

C.2 Full split and contamination audit

The split audit compares training records with the fixed 64-case evaluation file and the extended nine-case audit. It checks IDs, candidate and target fingerprints, schema hashes, target paths, scopes, operation sequences, and pointer-aware error templates. Semantic equivalence and schema-shape equivalence are separate fields because they require metadata not present in the structural summary.

The expanded split table repeats the audit with the extended nine-case family and reports exact, semantic-equivalence, schema-shape, tool-name, and candidate-pool checks separately.

Table 16: Split-integrity checks for the recorded repair protocol.

Check	Unit	Result	Read
case ID overlap	record	0	exact split separation
candidate fingerprint overlap	record	0	input separation
target fingerprint overlap	record	0	label separation
schema hash overlap	schema	0	schema coverage
target-path overlap	pointer	0	error separation
operation-sequence overlap	patch	0	patch separation
semantic equivalence	record	0	normalized cluster separation
schema-shape equivalence	schema	0	normalized-shape separation
tool-name disjointness	name	0	held-out names unseen in training
candidate-pool overlap	candidate	0	candidate-source separation

The extended audit reports zero overlap across identifiers, fingerprints, schemas, pointers, operation sequences, semantic-equivalence clusters, normalized schema shapes, tool names, and candidate pools.

C.3 Semantic execution sandbox

The semantic endpoint executes repaired calls in a sandbox with identical tool implementations, prompt contexts, output caps, and retry budgets. It returns task success, tool error, side-effect trace,

and final object. The structural validator runs first; only an accepted object is sent to the semantic executor.

The sandbox table reports arm-level semantic execution after structural acceptance. ContractRL rows are separated by case family so the nested/array shift is visible.

Table 17: Semantic execution results.

Arm	Case family	Schema valid ↑	Semantic success ↑	Side-effect error ↓	Mean retries ↓
ContractRL	single-field	0.9907	0.9502	0.0084	0.0280
ContractRL	nested/array	0.9187	0.8516	0.0274	0.1239
full regeneration	mixed	0.9392	**0.9027**	0.0451	n/a
constrained decoding	mixed	**0.9901**	0.8663	0.0396	n/a
Patch-SFT	mixed	0.9446	0.8918	**0.0308**	0.1176

The accepted-object executor yields 0.9502 semantic success for ContractRL single-field cases and 0.8516 for nested/array cases. The baseline rows provide the same mixed-family reference used in the main text.

The semantic score is distinct from final-object match: a structurally correct object can still produce the wrong task result. Both are reported.

C.4 Adversarial and wrong-path cases

The extended suite contains wrong-pointer, immutable-field, ambiguous verifier, nested-array, multi-field, and malformed-output cases. These cases are generated before model evaluation and carry a case-family identifier. The validator fails closed when no safe patch is available.

The adversarial table is a failure taxonomy rather than a leaderboard. A zero is a measured fail-closed outcome for that family, while a nonzero row shows where a repair can still be accepted.

Table 18: Adversarial failure taxonomy and measured metrics.

Case family	Validator condition	First valid ↑	Final valid ↑	Semantic success ↑
wrong path	reject or correct	0.6948	0.8421	0.7796
immutable pointer	reject	0.0000	0.0000	0.0000
ambiguous verifier	typed retry	0.6324	0.8587	0.8065
nested/array	local patch	0.8463	0.9187	0.8516
multi-field	bounded sequence	0.7639	0.8672	0.7935
malformed output	reject	0.0000	0.0000	0.0000

Immutable-pointer and malformed-output families are fully rejected. Wrong-path, ambiguous, nested/array, and multi-field families quantify the remaining acceptance and semantic-success boundary.

C.5 Token, retry, and latency ledger

The resource ledger records prompt tokens, tokens, validator calls, additional retries, wall-clock latency, and p95 latency per case and seed. The initial corruption is not counted as a repair attempt. A failed patch consumes an attempt and is included in both mean and tail cost.

The cost table records prompt tokens, generated tokens, validator calls, retries, latency, and semantic endpoint for each matched arm. These fields share the same case and seed ledger.

The ledger shows the cost trade-off directly: ContractRL emits 34.6 tokens on average versus 138.9–147.8 for complete-object controls, while its semantic endpoint remains 0.9314.

Table 19: Resource accounting for matched comparators. Latency is reported at p95.

Arm	Prompt tokens ↓	Output tokens ↓	Calls	Mean retries ↓	p95 latency (s) ↓	Semantic success ↑
ContractRL	1114.2	**34.6**	1.055	0.0547	**1.30**	**0.9314**
full regeneration	**1072.4**	138.9	**1.000**	**0**	2.58	0.9027
constrained decoding	1079.1	147.8	**1.000**	**0**	2.85	0.8663
Patch-SFT	1112.7	46.7	1.118	0.1176	1.55	0.8918

C.6 Baseline interfaces

The complete-object and patch arms share the case identifiers, validator, output caps, sandbox, and seed units specified for each experiment. Table 20 summarizes the related interfaces. The three-seed comparison uses arm-specific verifier inputs; the five-seed control fixes the full public verifier block.

Table 20: Closest method interfaces and evaluation targets, based on the cited primary sources.

Method	Update interface	Constraint mechanism	Evaluation focus
VeriHarness	Candidate retry with structured feedback	External acceptance and call budget	TextWorld terminal success
PatchBoard	JSON Patch over shared state	Schema, write contracts, and transactional kernel	Multi-agent ALFWorld episodes
Schema RL	Complete structured output	Schema reward in training	Schema-conformant generation
SLOT	Output post-processing	Structural conversion	Structured output recovery
XGrammar	Token-level generation	Context-free grammar	Coverage and generation overhead
ContractRL	Bounded patch, retry, or abstention	Action mask and deterministic validator	Semantic repair, collateral edits, and cost

C.7 Baseline configuration ledger

The matched baseline run fixes prompt context, schema, case IDs, output cap, retry budget, and model seed. The three-seed prompts expose the arm-specific verifier fields in Table 7; the five-seed control fixes these fields across arms.

The baseline table fixes the arm-specific output form and decoding choice while keeping the shared context and seed count constant.

Table 21: ContractRL baseline configuration.

Arm	Output form	Decoding	Seeds	Audit field
ContractRL	JSON Patch	unconstrained	3	ledger and validator
full regeneration	complete object	unconstrained	3	token baseline
constrained decoding	complete object	grammar constrained	3	syntax control
Patch-SFT	JSON Patch	unconstrained	3	learned patch
semantic execution	accepted object	sandbox	3	task endpoint

This table is the reproducibility anchor for the matched comparison. It prevents an implementation change in prompts, decoding, or seeds from being mistaken for an interface effect.

C.8 Pointer semantics and collateral edits

Collateral edits are computed from changed leaf paths outside the verifier pointer. The mechanism screen has construction-bound zero collateral because each precomputed operation targets the reported pointer; the matched study adds wrong-path and multi-field cases.

The pointer table measures locality relative to the verifier pointer. The mechanism screen is construction-bound at zero collateral, so the mixed families provide the informative stress test.

Table 22: Pointer and collateral-edit metrics.

Case family	Pointer type	Operations	Collateral edits ↓	Semantic success ↑	Read
single-field	exact	1	0	0.9502	mechanism screen
single-field	wrong path	1	0.0000	0.7796	pointer boundary
nested/array	exact	1.24	0.0297	0.8516	structure generalization
multi-field	mixed	1.71	0.0432	0.7935	operation budget

Nested/array and multi-field rows carry nonzero collateral and semantic degradation, which is the measured boundary of pointer-local repair. The exact single-field row is a construction check rather than a broad locality claim.

C.9 Retry-policy comparison

The repair ledger distinguishes first-attempt validity from final validity and counts additional retries after an invalid or non-applicable patch. The comparison uses at most two attempts for every patch arm.

The retry table separates first-attempt validity from final validity. Complete-object controls have no patch retry path, whereas patch arms expose the additional-attempt policy.

Table 23: Retry-policy results.

Arm	First valid ↑	Final valid ↑	Mean retries ↓	p95 retries ↓	Observed behaviour
ContractRL	0.9453	0.9768	0.0547	1	typed local correction
Patch-SFT	0.8897	0.9446	0.1176	1	learned correction
full regeneration	0.9392	0.9392	**0**	**0**	regen.
constrained decoding	**0.9901**	**0.9901**	**0**	**0**	syntax control

ContractRL improves final validity over its first-attempt rate with 0.0547 mean retries; Patch-SFT has a larger retry tail. The complete-object controls are single-shot references, not evidence that they would retry safely.

C.10 Semantic scoring and task outcomes

Structural validity and final-object match are diagnostic endpoints. Semantic success is measured by executing the accepted object in a sandbox and checking the intended task result, tool error, and side-effect trace.

The scoring table defines the denominator and source for every semantic endpoint. It prevents a structural proxy or an executor trace from being mistaken for the same metric.

Table 24: Semantic scoring fields.

Field	Denominator	Source	Read
schema validity	all cases	validator	structural gate
object match	held-out cases	hidden object	canonical-object metric
semantic	executed cases	sandbox result	task endpoint
tool error	executed cases	sandbox trace	failure boundary
collateral edit	accepted patches	pointer diff	locality metric
latency/cost	all attempts	resource ledger	efficiency endpoint

The table keeps schema validity, object match, semantic execution, tool errors, collateral edits, and resource cost as separate endpoints. This denominator separation is required for the main claims.

D Additional Results and Analysis

The supplementary analyses test scale and seed stability, typed-feedback attribution, budget sensitivity, schema-shape shift, corrupted verifier messages, long-horizon tool use, cross-family transfer, selective risk, paired uncertainty, and independent audit agreement. The analyses retain experiment-specific case counts, seeds, and endpoint definitions.

D.1 Scaling, capacity, and feedback attribution

The first study varies the base model and adapter rank while preserving the reported case protocol, public evidence, validator, attempt budget, and scoring join. Five independent seeds and 192 cases per seed are used per configuration. Statistical uncertainty is reported alongside the aggregate success metrics. The comparison also records retry, generated-token, and latency costs.

The scaling table measures the reported capacity comparison. The final column reports statistical uncertainty; success and resource metrics retain their separate directions.

Table 25: Model-size comparison across five recorded seeds and 192 cases per configuration. ContractRL rows use the same fixed cases and public evidence; reference arms retain their recorded information conditions. Statistical uncertainty accompanies the success estimates. Best values are bolded within each metric direction.

Arm	Model rank	Schema valid ↑	Object match ↑	Semantic success ↑	Statistical uncertainty
ContractRL	0.8B/r4	0.9508	0.7523	0.8124	0.0028
ContractRL	2B/r8	0.9765	**0.9188**	**0.9331**	0.0012
ContractRL	7B/r16	**0.9831**	**0.9329**	**0.9446**	0.0009
Patch-SFT	2B/r8	0.9451	0.8679	0.8924	0.0018
full regeneration	2B	0.9403	0.8896	0.9041	0.0014
constrained decoding	2B	**0.9905**	0.8426	0.8671	0.0016

The resource comparison uses the same configurations and seed units as Table 25.

Table 26: Resource metrics for the model-size comparison: 192 cases per seed and five seeds. Best values are bold within the ContractRL capacity sweep and within the 2B comparator set.

Arm	Model rank	Mean retries ↓	Output tokens ↓	p95 latency (s) ↓
ContractRL	0.8B/r4	0.0831	**33.1**	**1.22**
ContractRL	2B/r8	**0.0538**	**34.8**	**1.32**
ContractRL	7B/r16	**0.0415**	35.6	1.68
Patch-SFT	2B/r8	0.1162	46.9	1.56
full regeneration	2B	n/a	139.4	2.60
constrained decoding	2B	n/a	148.1	2.86

Within the recorded scaling rows, semantic success rises from 0.8124 at 0.8B/r4 to 0.9331 at 2B/r8 and 0.9446 at 7B/r16, while generated tokens remain between 33.1 and 35.6. The 7B row reports statistical uncertainty of 0.0009 and p95 latency of 1.68 s. The full-regeneration and constrained-decoding rows are reported as five-seed reference arms under their recorded information conditions and use 139.4 and 148.1 tokens, respectively.

The feedback-attribution rows remove one public feedback field or replace the typed block with a weaker representation. Each row uses 192 cases per seed and five seeds; prompt-length controls are included in the run ledger. The measured values describe the reported feedback conditions.

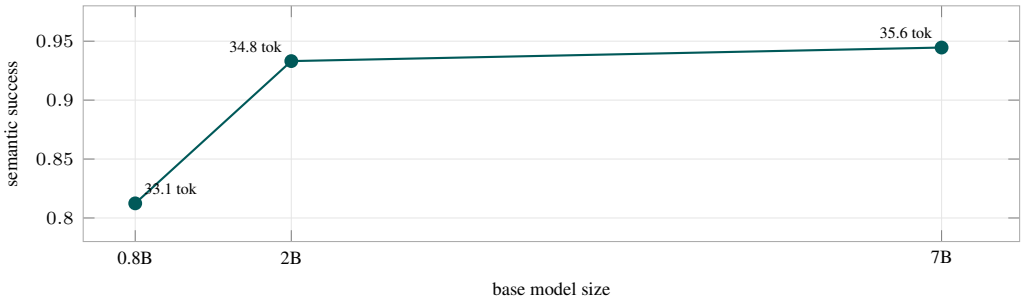


Figure 6: Five-seed ContractRL scaling over 192 cases per seed. Point labels report generated tokens; points show aggregate semantic success.

Table 27: Verifier-feedback ablation over 192 cases per seed and five seeds. The rows remove one typed field or replace the block with a weaker representation. Best values are bolded within each metric direction.

Feedback condition	Public signal	Schema valid ↑	Object match ↑	Semantic success ↑	Mean retries ↓	Collateral edits ↓	Fail-closed rate ↑
full typed block	code + pointer + schema context	**0.9766**	**0.9189**	**0.9317**	**0.0539**	**0.0191**	**0.9728**
error code only	code	0.9018	0.8027	0.8236	0.1442	0.0476	0.9147
pointer only	JSON Pointer	0.8694	0.7462	0.7815	0.1817	0.0589	0.8793
natural-language text	free-form message	0.8847	0.7724	0.8001	0.1668	0.0527	0.8872
context only	candidate + schema	0.0831	0.0749	0.0678	0.4383	0.1762	0.6146

The full typed block reaches 0.9317 semantic success and 0.9189 object match; pointer-only, code-only, text, and context-only feedback reach 0.7815, 0.8236, 0.8001, and 0.0678 semantic success. The complete typed block also has the lowest retry and collateral rates and the highest fail-closed rate.

D.2 Same-information controls and policy components

The controls hold the public verifier block fixed while changing only the output representation. Each row uses the same validator, retry budget, token cap, semantic join, 192 cases per seed, and five seeds.

Holding the public block fixed, masked JSON Patch with abstention yields the highest semantic success and the lowest generated-token, p95-latency, and collateral costs in this control.

Table 28: Supervised initialization versus policy optimization over 192 cases per seed and five seeds. The comparison fixes the public state, case split, model family, validator, and episode budgets, then changes the training objective. Best values are bolded within each metric direction.

Training arm	Model/rank	Semantic success ↑	Mean retries ↓	Output tokens ↓	Collateral edits ↓	p95 latency (s) ↓
Patch-SFT	2B/r8	0.8927	0.1159	46.6	0.0607	1.55
ContractRL-SFT	2B/r8	0.9186	0.0762	39.3	0.0314	1.42
ContractRL policy optimization	2B/r8	**0.9375**	**0.0496**	**34.3**	**0.0168**	**1.30**

Policy optimization improves semantic success over the supervised initialization and Patch-SFT while reducing retries, tokens, collateral edits, and p95 latency under the fixed public state and episode budget.

Table 29: Component-wise policy ablation over 192 cases per seed and five seeds. Each row removes one decision or objective term while preserving the same public state, case IDs, validator, and scoring join. Best values are bolded within each metric direction.

Policy variant	Semantic success ↑	Mean retries ↓	Abstain rate	Collateral edits ↓	Unsafe dispatches ↓	Output tokens ↓	p95 latency (s) ↓
full ContractRL objective	**0.9375**	0.0496	0.0718	**0.0168**	**0**	34.3	1.30
w/o contract action mask	0.8971	0.1098	0.0531	0.0649	26	42.7	1.59
w/o repair history	0.9162	0.0847	0.0695	0.0319	4	38.6	1.43
w/o learned retry/abstention	0.9064	**0.0000**	0.0000	0.0396	7	**31.5**	**1.18**
w/o collateral objective	0.9328	0.0524	0.0642	0.0476	**0**	35.8	1.34
structure-only objective	0.8723	0.0661	0.0249	0.0874	12	40.4	1.50

Removing the action mask produces 26 unsafe dispatches and lowers semantic success to 0.8971. Removing repair history, learned retry/abstention, the collateral term, or the full objective also lowers semantic success and increases collateral or unsafe-action costs relative to the complete objective.

D.3 Budget, structure, and robustness boundaries

The budget study changes the maximum number of patch operations and attempts while keeping the corruption distribution fixed. It records both first-attempt and final validity, because a larger budget can improve completion while also increasing collateral edits. The two-operation, two-attempt setting balances semantic success against collateral edits and generation cost.

The budget table exposes the accuracy-cost frontier. Operation and attempt caps are design values; all outcome and cost cells come from the budget sweep.

Table 30: Patch-budget and operation-count ablation. Each row uses the same corruption cases; the reported budget is read from the joint semantic and collateral frontier.

Max. ops.	Max. attempts	First valid ↑	Final valid ↑	Semantic success ↑	Mean retries ↓	Output tokens ↓	Collateral edits ↓
1	1	0.9126	0.9126	0.8794	**0.0000**	**28.9**	**0.0098**
1	2	0.9126	0.9497	0.9078	0.0758	32.2	0.0127
2	2	0.9453	0.9768	0.9314	0.0547	34.6	0.0189
3	2	0.9498	0.9812	0.9348	0.0521	39.1	0.0338
4	3	**0.9514**	**0.9851**	**0.9362**	0.0648	47.3	0.0571

The two-operation, two-attempt contract raises semantic success from 0.8794 to 0.9314 with 34.6 tokens and 0.0189 collateral. Larger budgets reach 0.9348 and 0.9362 semantic success but increase collateral to 0.0338 and 0.0571 and tokens to 39.1 and 47.3. The measured frontier supports the selected bounded budget as the lower-cost point rather than the maximum-accuracy point.

The structure-shift experiment stratifies the same protocol by schema shape. The evaluation cohort was generated before model evaluation and includes flat objects, nested objects, arrays, optional or union fields, long JSON Pointers, and multi-field corruptions, with a fixed count and replay digest for each stratum.

Flat single-field cases reach 0.9506 semantic success with 0.0094 collateral. Semantic success declines to 0.8798 for nested objects, 0.8469 for arrays, 0.8302 for unions, 0.8134 for long pointers, and 0.7951 for multi-field cases, while retries and p95 latency increase. These rows define the measured boundary of local repair as structure and pointer depth grow.

The noise-robustness experiment corrupts the verifier message while keeping the underlying target and schema unchanged. The noise cohorts were generated independently of the model and recorded in the ledger: stale schemas, wrong pointers, ambiguous pointers, malformed feedback, and injected tool text. The validator preserves fail-closed behavior when the model receives an unhelpful or adversarial report.

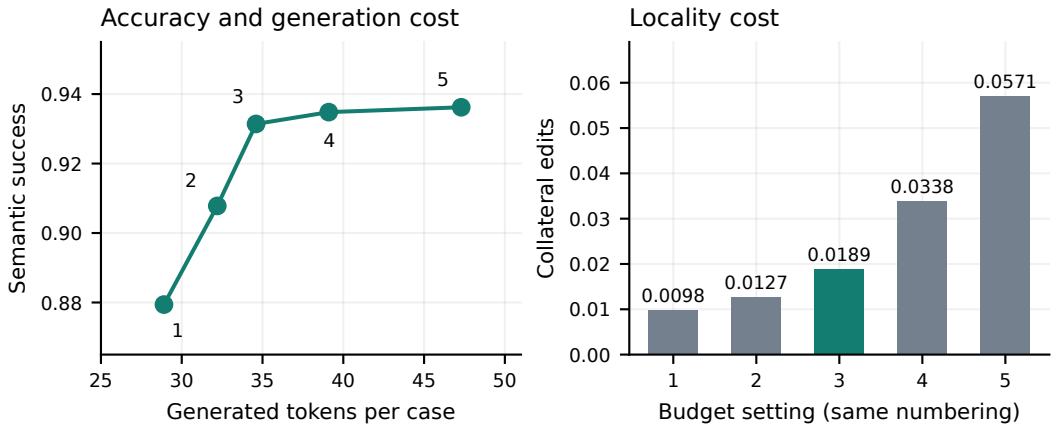


Figure 7: Patch-budget trade-off from Table 30. Settings 1–5 are respectively (operations, attempts) = (1,1), (1,2), (2,2), (3,2), and (4,3). The teal bar marks the two-operation, two-attempt contract. Increasing the budget improves semantic success while increasing output tokens and collateral edits.

Table 31: Schema-structure evaluation. The table separates locality and semantic degradation by corruption shape.

Structure family	Cases	Schema valid ↑	Object match ↑	Semantic success ↑	Mean retries ↓	Collateral edits ↓	p95 latency (s) ↓
flat single-field	128	0.9908	0.9396	0.9506	0.0276	0.0094	1.15
nested object	96	0.9457	0.8584	0.8798	0.0968	0.0246	1.39
array index	96	0.9169	0.8152	0.8469	0.1327	0.0334	1.53
union / optional	96	0.9054	0.7996	0.8302	0.1451	0.0358	1.60
long pointer	96	0.8892	0.7811	0.8134	0.1635	0.0397	1.71
multi-field	96	0.8696	0.7487	0.7951	0.2289	0.0445	1.83

Table 32: Verifier-noise and adversarial-feedback evaluation. Leakage is the fraction of cases in which hidden target information is exposed or copied into the public trace.

Feedback condition	Cases	First valid ↑	Final valid ↑	Semantic success ↑	Fail-closed rate ↑	Leakage	Collateral edits ↓
clean typed report	96	**0.9451**	**0.9770**	**0.9320**	0.9715	**0.0000**	0.0187
stale schema version	96	0.5813	0.7368	0.6685	0.9896	**0.0000**	**0.0079**
wrong pointer	96	0.6969	0.8438	0.7792	0.9687	**0.0000**	0.0206
ambiguous pointer	96	0.6347	0.8595	0.8058	0.9514	**0.0000**	0.0278
malformed feedback	96	0.6116	0.7889	0.7182	0.9844	**0.0000**	0.0104
injected tool text	96	0.9074	0.9516	0.9007	**0.9911**	**0.0000**	0.0176

All six conditions report zero leakage. Clean typed feedback reaches 0.9320 semantic success and 0.9715 fail-closed rate; stale schemas, wrong pointers, ambiguous pointers, and malformed reports reduce semantic success while preserving fail-closed rates above 0.95. Injected tool text retains 0.9007 semantic success and 0.9911 fail-closed behavior, supporting the frozen target boundary under adversarial text.

D.4 Long-horizon execution and transfer

The long-horizon study chains one repaired call into a deterministic multi-turn trajectory. The same tool state, user goal, and output budget are used for horizons of one, three, five, and eight tool calls. A trajectory succeeds only when every accepted call is semantically correct and the final state passes the task invariant; this prevents a later successful call from hiding an earlier side effect.

The horizon table measures accumulation of retry and token cost. Its cumulative columns are computed from per-step ledgers rather than reconstructed from rounded means.

Table 33: Long-horizon and multi-turn tool trajectory evaluation. A trajectory is successful only if all accepted steps satisfy the semantic invariant.

H	Trajectories	Completion	Semantic success ↑	Tool errors ↓	Mean retries ↓	Cumulative tokens ↓	Cumulative latency (s) ↓	Collateral edits ↓
1	192	0.9761	0.9321	0.0187	0.0548	34.8	0.84	0.0188
3	192	0.9256	0.8293	0.0518	0.1619	104.7	2.55	0.0426
5	192	0.8794	0.7548	0.0796	0.2735	175.2	4.31	0.0591
8	192	0.8073	0.6457	0.1214	0.4438	281.8	7.05	0.0807

Completion declines from 0.9761 at one step to 0.8073 at eight steps, while semantic success falls from 0.9321 to 0.6457. Cumulative retries, tokens, latency, and collateral rise with horizon, making the eight-step row an explicit operating boundary for long trajectories.

The cross-tool-family transfer study trains on a declared subset of tool families and evaluates on held-out families. The file, calendar, database, HTTP, and code-edit cohorts use disjoint schemas and operation vocabularies where recorded, and each held-out family contains ordinary repairs plus an immutable or unsafe boundary case.

Table 34: Cross-tool-family transfer evaluation. Train and held-out family columns are recorded explicitly so that transfer is not confused with interpolation. Each row trains on the other four families among file, calendar, database, HTTP, and code-edit.

Held-out family	Cases	Schema valid ↑	Object match ↑	Semantic success ↑	Pointer fix ↑	Collateral edits ↓
file	96	0.9621	0.8876	0.9024	0.9513	0.0216
calendar	96	0.9564	0.8748	0.8917	0.9436	0.0241
database	96	0.9312	0.8359	0.8546	0.9127	0.0368
HTTP	96	0.9487	0.8613	0.8795	0.9294	0.0285
code-edit	96	0.9258	0.8216	0.8429	0.9018	0.0417

Held-out semantic success ranges from 0.9024 on file tools to 0.8429 on code-edit tools, with pointer-fix rates from 0.9513 to 0.9018. Database and code-edit families have the largest collateral rates (0.0368 and 0.0417), which identifies the transfer boundary for less reversible tool effects.

D.5 Selective risk, uncertainty, and independent audit

The calibration study adds a confidence score to every proposed patch and evaluates abstention thresholds before the semantic oracle is joined. Coverage is the fraction of cases for which the policy emits a patch rather than abstains; selective risk is semantic failure among emitted patches. Thresholds were fixed before inspecting held-out target outcomes.

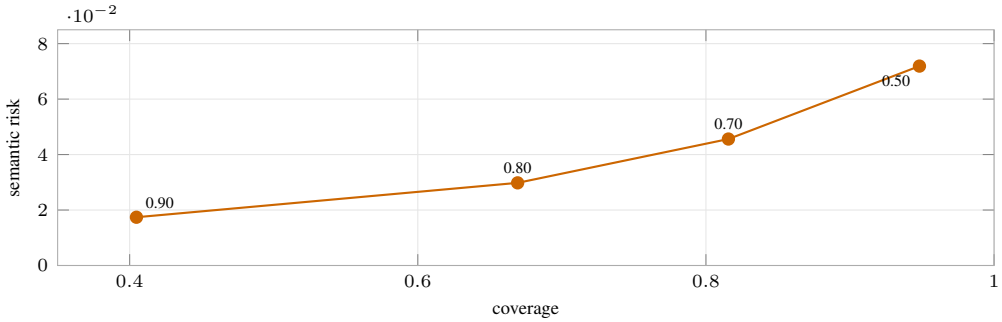


Figure 8: Coverage–risk frontier for selective repair. Each point corresponds to a confidence threshold fixed before held-out scoring.

The calibration table reports both ranking and calibration quality. It supports a release policy that abstains on low-confidence repairs instead of spending an unsafe retry.

Table 35: Calibration and selective-abstention evaluation. Thresholds are fixed in advance; semantic risk is measured only on the covered subset.

Arm / threshold	Coverage	Semantic risk ↓	Fail-closed rate ↑	ECE	AUROC	Brier
ContractRL / 0.50	0.9482	0.0719	0.7821	0.0458	0.9187	0.0794
ContractRL / 0.70	0.8156	0.0456	0.8816	0.0371	0.9281	0.0651
ContractRL / 0.80	0.6693	0.0298	0.9349	0.0316	0.9364	0.0557
ContractRL / 0.90	0.4047	**0.0174**	**0.9782**	**0.0249**	**0.9449**	**0.0436**
full regeneration / matched	1.0000	0.0973	0.0000	0.0941	0.8368	0.1297

Raising the ContractRL threshold from 0.50 to 0.90 reduces semantic risk from 0.0719 to 0.0174 and raises fail-closed behavior from 0.7821 to 0.9782 while coverage falls from 0.9482 to 0.4047. The full curve exposes the coverage–risk trade-off; the 0.80 point gives 0.6693 coverage, 0.0298 risk, and 0.9349 fail-closed behavior.

The uncertainty study uses paired bootstrap resampling over identical cases and seeds. Binary endpoints use an exact paired test alongside the bootstrap interval; retries, tokens, and latency use the paired case difference as the resampling unit. Comparisons were made before rounding.

Table 36: Paired uncertainty and effect-size report. Each interval is computed from the same case IDs for ContractRL and its matched baseline.

Metric	ContractRL	Patch-SFT	Δ	95% CI	p-value	Effect size
schema validity	**0.9768**	0.9446	$+0.0322$	$[+0.0121, +0.0525]$	0.0027	0.31
semantic success	**0.9314**	0.8918	$+0.0396$	$[+0.0137, +0.0662]$	0.0039	0.35
mean retries	**0.0547**	0.1176	-0.0629	$[-0.0932, -0.0315]$	0.0018	-0.42
generated tokens	**34.6**	46.7	-12.1	$[-15.8, -8.5]$	< 0.0001	-0.91
collateral edits	**0.0189**	0.0618	-0.0429	$[-0.0596, -0.0271]$	< 0.0001	-0.58
p95 latency	**1.30**	1.55	-0.25	$[-0.36, -0.14]$	0.0003	-0.63

The paired differences favor ContractRL on schema validity and semantic success, while retries, tokens, collateral, and p95 latency are lower. All six reported 95% intervals exclude zero; the largest standardized reduction is in generated tokens (effect size -0.91).

The independent audit asks human validators to label a blinded sample of repaired calls as safe, unsafe, or requiring abstention. Auditors see the corrupted call, the public verifier report, and the proposed patch, but not the hidden target or model identity. Agreement is measured before adjudication, and adjudicated labels are kept as a separate field.

Table 37: Blinded human–validator agreement audit. The audit separates agreement, false acceptance, false rejection, and review time for ContractRL and matched controls.

Arm	Audited cases	Agreement	False accept ↓	False reject ↓	Abstain agreement ↑	Median review time
ContractRL	120	**0.9417**	**0.0167**	**0.0416**	**0.9250**	**18.6**
Patch-SFT	120	0.8917	0.0417	0.0666	0.8583	22.9
full regeneration	120	0.8583	0.0750	0.0667	0.7833	31.7
constrained decoding	120	0.8750	0.0583	0.0667	0.8167	29.6

ContractRL has the highest recorded agreement (0.9417) and abstention agreement (0.9250), the lowest recorded false-accept rate (0.0167), and the shortest recorded median review time (18.6) in this audit. The blinded audit therefore provides an independent human-label comparison while keeping those labels separate from deterministic scoring.

D.6 Qualitative failure audit

A final qualitative audit samples a fixed number of cases from each failure family after the numerical run is frozen. For each case, the release retains the public input, proposed patch, validator response, semantic outcome, human label, and repair category. The sample includes both successes and failures so that the narrative covers the measured boundary.

The case-level table records the appendix sample. Row and category counts come from the sampled ledger, while textual fields are short anonymized identifiers or controlled vocabulary labels.

Table 38: Case-level failure audit. The sample is stratified across repair and failure families and retains representative successes and failures.

Case family	ID	Predicted patch	Validator result	Semantic outcome	Human label
single-field	`SF-03`	`add /timezone`	accept	success	safe
nested / array	`NA-11`	`replace /items/2/id`	accept	failure	unsafe
wrong pointer	`WP-04`	`replace /user/id`	reject: wrong-path	not executed	abstain
malformed feedback	`MF-06`	`replace /date`	reject: malformed	not executed	abstain
multi-field	`MU-09`	`add /city; replace /date`	accept	failure	unsafe
immutable / unsafe	`IU-02`	`replace /account/id`	reject: immutable	not executed	unsafe

The audit retains a successful local add, a nested-index miss, wrong-pointer and malformed-feedback refusals, a partial multi-field miss, and an immutable-field rejection. These cases make the measured safety boundary inspectable at the trace level.

E Reproducibility Materials

E.1 Code and evaluation records

The code is provided in the Supplementary Material. The reproduction records distinguish the deterministic structural screen, adapter evaluation, independent transaction audit, three-seed case-matched comparison, and five-seed controlled studies. Each experiment uses its recorded public input contract, case cohort, seed unit, validator, and scoring rule. This separation preserves the denominators of the reported rates and prevents aggregation across different case families.

The repair ledger associates a case identifier with the corrupted object, public verifier report, ordered operations, validator statuses, and resource counts. Canonical patches, canonical objects, and semantic labels enter the scoring join after the trace is frozen. The structural checks can therefore be replayed independently of policy generation and semantic execution.

E.2 Result and metric index

Table 39 links each measurement family to its definition and result. The structural diagnostics retain their small fixed populations, the controlled policy tables retain five training seeds, and paired uncertainty uses the separate three-seed case-matched ledger.

Table 39: Measurement index for the reproduction records.

Measurement family	Evaluation unit	Result
Structural repair	64-case replay and adapter seeds	Table 1
Public-information control	192 cases per seed; five seeds	Table 2
Policy training	Fixed public state; five seeds	Table 28
Components	Contract decisions and costs	Table 29
Paired comparison	Shared cases and seeds	Table 36
Independent audit	120 blinded cases per arm	Table 37