SPLICEGUARD: COMPOSITION-TIME CONTRACTS RECOVER STRICT INFERENCE BUDGETS

Anonymous authors
Paper under double-blind review

ABSTRACT

Terminal validation can detect a structural defect but cannot refund calls spent extending invalid intermediate state. SPLICEGUARD turns certificates into composition-time control: interface refinement governs admission, one bounded repair can restore a failed boundary, and dependency-aware replay recomputes only the affected suffix. Freeze-then-replay holds candidate membership, executor, cache policy, accounting, and replay order fixed to identify budget recovery. Under GPT-4o-mini (2024-07-18) and a 10k-call limit, replaying 38,482 candidates raises candidate-level final-checker accuracy from 72.3% to 82.7% with contract admission and to 85.6% with repair. A second 27,346-candidate replay compares terminal validation, eager checks, full-rerun schema gating, contract admission, and repair under the same instance-level success denominator: gate-only reaches 75.8% and gate+repair reaches 78.1%, versus 64.1%, 69.2%, and 71.3% for no filtering, eager validation, and a full-rerun schema gate. The comparison isolates refinement authority and replay locality beyond earlier detection. A structural refinement proposition and a fixed-pool throughput condition specify when early checking can convert avoided late work into usable budget. Composition-time contracts thereby make strict-budget workflow composition an auditable control mechanism.

1 INTRODUCTION

Hard inference budgets make validation timing a systems decision. A workflow can keep expanding a candidate after its type, schema, evidence, or tool interaction violates a deterministic requirement; terminal validation records the failure after the affected calls are spent. We study *late structural failure* as a budget-conversion problem: the same certificate can label waste at termination or prevent it at the composition boundary.

SPLICEGUARD implements this choice as a contract-governed composition operator. Each module exposes accepted inputs, produced outputs, deterministic predicates, and required evidence through $\mathcal{I} = (\mathcal{T}_{in}, \mathcal{T}_{out}, \Pi, \mathcal{E}^{\star})$. At a proposed splice, refinement selects a charged transition: admit and replay the affected suffix, reject before suffix execution, or apply one bounded repair and re-check the dependency closure. Contract authority, cache invalidation, and accounting meet at this event, distinguishing SPLICEGUARD from typed wiring, decode-time constraints, eager checks, and terminal scoring.

Table 1 connects these admission and repair outcomes to final-checker accuracy on a 38,482-candidate frozen pool spanning Spider, UT-Python, and BBH-Logic.

Identifying the budget effect requires fixed candidates and changing enforcement. We freeze the first pool before gate outcomes, then replay it with the same executor, cache policy, ordered ledger, and 10k-call budget under no filter, contract admission, and admission plus repair. A second 27,346-candidate pool adds terminal validation, eager node validation, and a splice schema gate with full rerun. The two controlled replays separate intervention timing, refinement authority, bounded repair, and replay locality from proposal membership.

The paper contributes:

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.

- A composition operator that unifies refinement-based admission, one bounded deterministic repair, dependency-closure invalidation, and suffix-local replay under a single charged budget.
- A matched-accounting identification protocol over two frozen proposal pools. Internal checker accuracy rises 72.3%→82.7%→85.6%; in the six-arm replay, SPLICEGUARD reaches 78.1% versus 69.2% for eager validation and 71.3% for schema gating with full rerun.
- A denominator-aware fixed-pool analysis that separates final task correctness, pre-repair boundary status, failure lateness, checking overhead, repair recovery, and suffix-replay cost.

The analysis proves splice-boundary refinement and derives a sufficient fixed-pool throughput threshold (Prop. 1). Across the controlled replays and the six-task protocol audit, the same principle emerges: certificates protect strict budgets when they control composition and its recomputation boundary.

2 RELATED WORK

The closest systems separate along five operational axes: constraint location, time of detection, repair authority, recomputation locality, and whether evaluation preserves a budget counterfactual. PICARD prunes inadmissible tokens during SQL decoding (Scholak et al., 2021); MRKL combines neural and symbolic modules, while DSPy exposes signatures for declarative pipelines (Karpas et al., 2022; Khattab et al., 2024). These systems constrain generation or wiring. SPLICEGUARD governs dynamic module replacement: interface refinement jointly determines splice admission, bounded repair, cache invalidation, and the suffix charged for replay.

ReAct interleaves reasoning and actions (Yao et al., 2023). Reflexion, Self-Refine, and learned verifiers use feedback or post-generation evaluation to improve trajectories; ToolVerifier self-checks tool selection and parameter generation (Shinn et al., 2023; Madaan et al., 2023; Mekala et al., 2024; Cobbe et al., 2021). Contract-governed composition gives deterministic checks authority before the proposed suffix executes and binds an accepted edit to dependency-closure replay. The distinction is economically consequential under a hard budget: post-trajectory feedback can improve the next attempt, whereas splice-time control preserves calls for the current search.

Language-agent tree search, AgentSwift, and EvoFlow allocate calls across trajectories or evolve workflow structure (Zhou et al., 2024; Li et al., 2025; Zhang et al., 2025). SPLICEGUARD adds a certificate-bearing composition transition with a shared spend ledger. Its second fixed-pool replay freezes candidate identity before comparing terminal checks, eager checks, full-rerun schema gates, contract admission, and repair. This matched replay exposes the timing-and-authority effect hidden by end-to-end comparisons. Proof-carrying code provides the precedent for machine-checkable evidence (Necula, 1997); SPLICEGUARD operationalizes that evidence as an inference-throughput control.

3 PROBLEM SETUP

We consider inference-time workflows with deterministic structural checks whose cost permits operator-level intervention. The target quantity is recovered usable throughput: final-checker-correct outputs per charged call when structural failures would otherwise surface late. Explicit types, schemas, evidence fields, and tool-I/O predicates enable invalid candidates to be stopped before they consume the remaining workflow budget.

Our task scope requires an automated, deterministic assessment of final correctness, for example through unit tests, SQL execution, or exact match. Reasoning procedures take the form of typed operator DAGs (PROPOSE, VERIFY, REFLECT, AGGREGATE, TOOLCALL); a context-free graph grammar generates them with acyclic structure and type-safe connections. Executability thus follows from the grammar by construction; certificates govern the actions following local edits or recombination.

Node outputs pair an artifact with a local certificate containing machine-checkable evidence and deterministic predicates. Failure of any required predicate for types, schemas, evidence, or tool I/O makes a candidate *structurally invalid*. Search uses (i) the certificate verifier $\mathrm{CertCheck}(\tau)$; final evaluation alone uses (ii) the distinct external task checker $\mathsf{Check}(x, \hat{y})$. We quantify reliability by *false accepts*, where certificate validity accompanies a wrong checker outcome, and *false rejects*,

where certificate invalidity accompanies a correct checker outcome; false accepts threaten gating reliability, while false rejects discard checker-correct candidates and can reduce usable throughput.

The controlled comparisons treat the ordered proposal stream as an exogenous input. Candidate membership is frozen before policy outcomes are observed, and the external task checker is used only to label the policy-specific final output. The paper makes no claim about using checker outcomes, certificate scores, archive replacement, or tie-breaking to select an element of either frozen pool.

For a fixed budget B, denote the structural invalid-attempt rate and failure lateness without gating by (ρ, d) and their gated equivalents by (ρ', d'). Let o measure verification expense in units of a full evaluation, r the fraction of boundary-failing candidates recovered by bounded repair, and q the cost of rerunning only downstream nodes, also relative to a full evaluation. Write C for the full-evaluation cost and π for final-task correctness probability conditional on structural validity. Throughput without gating equals $\mathcal{T}_0 = \pi(1-\rho)/\big(C((1-\rho)+\rho d)\big)$; adding certificate gating and bounded repair gives

$$\mathcal{T}(\rho', d', o, r, q) \;=\; \frac{\pi \cdot \big((1-\rho') + \rho' r\big)}{C \cdot \big(o + (1-\rho') + \rho' d' + \rho' r q\big)}, \qquad \Delta^\star \;=\; B\big(\mathcal{T} - \mathcal{T}_0\big).$$

A replay begins with ledger $b = B$. Before an executor, mutator, repair, re-execution, or tool operation, its charge is debited from the same ledger; an operation that would exceed the remaining ledger is not executed and terminates that task replay. A frozen pool fixes the candidate identities and their replay order before policy outcomes. A candidate is counted in the fixed-pool denominator whether or not its policy-specific suffix is executed. Saved suffix calls remain in the ledger and can be spent only on later entries in that fixed order. Each candidate produces a policy-specific final outcome: an unrepaired rejection contributes zero final correctness after its proposal and check charges, while a repaired-and-admitted candidate is evaluated by the external task checker. The reported experiments share cache keys and cache granularity across arms; the manuscript does not infer unrecorded generation-level randomness coupling.

4 CONTRACT-GOVERNED WORKFLOW COMPOSITION

SPLICEGUARD integrates admission, repair, and replay into a single composition operator. A deterministic certificate can veto an incompatible splice before suffix execution, authorize one bounded repair episode, or admit the replacement and invalidate precisely its downstream dependency closure. Every check, edit, and re-executed call enters the same budget ledger. The resulting technical object is a certificate-bearing state transition that controls both workflow structure and the spend induced by changing it.

We represent a module-boundary interface by $\mathcal{I} = (\mathcal{T}_{in}, \mathcal{T}_{out}, \Pi, \mathcal{E}^\star)$: $\mathcal{T}_{in/out}$ denote the input/output types, Π collects the deterministic predicates that must hold, and $\mathcal{E}^\star$ identifies evidence fields that must be logged. As certificate-carrying interfaces, certificates restrict how modules compose, initiate repair upon boundary failure, and record each decision in an audit trail.

A spliced boundary meets its required interface *by construction* if the replacement module meets its certified interface and the latter is at least as strong as the requirement: its inputs are broader, its outputs narrower, and its required predicates and evidence fields form a superset. Only that boundary receives this guarantee; it does not extend to the entire offspring. Every accepted splice or repair therefore still requires SPLICEGUARD to rerun affected downstream nodes and check certificates again. Appendix A.4 provides the definition of refinement and its formal proof.

Let state $S = (G, \tau, \mathcal{L}, b)$ contain the current DAG, trace, audit ledger, and remaining budget. For proposed replacement m at boundary v, let a mean that refinement and the applicable boundary predicates pass, and f mean that a failed boundary is eligible for bounded repair. The transition is

$$S \xrightarrow{\text{splice}(v,m)} \begin{cases} (G, \tau, \mathcal{L} \cup \{\textsc{Reject}\},\, b - oC), & \neg a \wedge \neg f, \\ \text{Replay}_{\downarrow v}(\text{Repair}(G, m)), & \neg a \wedge f, \\ \text{Replay}_{\downarrow v}(G[v \leftarrow m]), & a. \end{cases}$$

The first branch spends verification cost oC and prevents downstream execution. The remaining branches charge deterministic repair when invoked and replay only nodes reachable from v, with every affected certificate checked again. This transition exposes the exact operational difference from terminal validation and eager checks that lack refinement authority or suffix-local cache semantics.

Two sufficient conditions turn measured invalidity, lateness, repair, and verifier error into a deployment decision. Fix a proposal pool containing N candidates and a budget B. In the no-gate mode, (ρ, d, o) denote, respectively, invalid-attempt rate, lateness of failure, and verification overhead. For gated operation use (ρ', d', r, q), subject to $\rho' < \rho$ and $d' \leq d$; bounded repair recovers a fraction $r \in [0, 1]$ of the ρ' boundary-failing candidates, with cost $q < 1$ for downstream-only re-execution. Applying the throughput formula from Section 3 yields:

Proposition 1 (Sufficient fixed-pool throughput condition)**.** *If* $o < \rho d - \rho'(1 - r)d' - \rho' rq$*, then* $\mathcal{T}(\rho', d', o, r, q) \geq \mathcal{T}_0(\rho, d)$ *with strict inequality whenever* $\rho > \rho'$ or $d' < d$*.*

The threshold states the actionable regime: avoided late-failure spend plus recovered children must exceed checking and suffix-replay cost. It is a conditional fixed-pool statement; Appendix A.4 gives the direct proof. The reported pre-repair boundary-status cells are descriptive statistics, not substitutions for conditional verifier-error probabilities in this proposition.

From parent DAGs, SPLICEGUARD generates offspring using (i) mutations constrained by GED bounds and the grammar and (ii) crossover checked against contracts: sub-DAG modules may be exchanged only if contracts match or refine at the boundary being spliced. Following either an edit or a splice, re-execution is limited to affected downstream nodes, and certificate checks to those impacted. A fixed taxonomy (schema, type, tool, evidence, cascade) classifies logged offspring rejections, allowing recombination behavior to be measured rather than described anecdotally.

A failed verification permits bounded repair with a limit of K steps. Fixed templates determine repairs deterministically; they are not learned. Each step incurs a budget charge and a log entry recording artifacts before and after repair, changes in predicates, added cost, and nodes executed again. An offspring that remains verification-invalid after K steps is rejected.

The contract-induction path derives $\widehat{\mathcal{I}}$ on the *train* split from tool schemas that supply Level-0 predicates for types and fields, outputs of harness validators that supply deterministic predicates checked after execution, and recurring failures mined from a set of 512 no-gate trajectories drawn exclusively from the training split. This deterministic path uses no LLM calls.

SPLICEGUARD loads $\mathcal{I}$ (manual or induced) and applies contract-checked crossover or GED-bounded mutation to a proposed candidate. It executes with shared caching and invokes at most one repair episode of $K = 3$ deterministic steps before acceptance or rejection. Every executor, mutator, repair, replay, and tool call counts under one global budget, and all compared methods use the same cache key and granularity. Algorithm 1 specifies the fixed-pool policy-evaluation loop.

5 EXPERIMENTS

The evaluation answers three scientific questions: does composition-time enforcement recover calls that terminal checks lose; does bounded repair improve on rejection alone; and do refinement authority and local replay improve on timing-only or full-rerun controls? Table 1 isolates admission and repair in one frozen pool, Table 2 compares six validation operators in a second fixed pool, and Appendix A.2.2 reports the verifier-available six-task protocol view.

Each task replay has a single global allowance of 10,000 calls, charged for all executor, mutator, repair, re-execution, and tool calls. Before a call is made, its charge is assessed against the remaining ledger; a policy cannot spend beyond that ledger. The task checker labels final outputs and never selects a candidate from a frozen pool. All controlled runs use Python 3.11 on 8×NVIDIA A100-80GB with CUDA 12.4 driver 550.54.14; the shared executor is GPT-4o-mini (2024-07-18) with $T = 0.7$ and top-$p = 0.95$. We report task success, candidate-level structural outcomes, charged-call fractions, API cost, and failure lateness with each denominator stated in the associated table.

The paper reports six verifier-available task settings: BBH-Logic, Boolean Expressions, and Dyck from BBH (Suzgun et al., 2022); UT-Python; GSM-Hard (Gao et al., 2023); and Spider (Yu et al., 2018). Appendix A.2.2 gives the exact split label and task-instance count used in each row. The two fixed-pool replays use Spider, UT-Python, BBH-Logic, BIRD dev (Li et al., 2023), and Tool-Sandbox (Lu et al., 2024), as named in Tables A.1 and 2; those tables, rather than a portability claim, define the reported evaluation units. All controlled arms share the executor, tools, cache policy, and accounting described in Appendix A.1.1.

For the first timing audit, we run one shared proposal generator on Spider, UT-Python, and BBH-Logic, freeze 38,482 candidates before intervention outcomes, and replay identical membership under no-filter, gate-only, and gate+repair. Rejection charges proposal and checking calls but skips the unexecuted suffix; repair charges its deterministic edits and affected-suffix replay.

The second audit freezes 27,346 candidates before replay: 14,728 from BIRD dev and 12,618 from ToolSandbox, over 2,566 held-out instances (1,534 BIRD; 1,032 ToolSandbox). Six arms share ten seeds, a 10k-call ledger, cache keys, and candidate order: no filter; terminal validation; eager node-output validation without refinement or repair; a splice schema gate followed by full downstream rerun; SPLICEGUARD gate-only; and SPLICEGUARD gate+repair. Task success is computed per instance; candidate-level validity, failure lateness, wasted-late calls, verification overhead, pre-repair boundary-status cells, repair recovery, suffix-replay fraction, and API cost use the frozen-candidate denominator. Confidence intervals use a paired bootstrap clustered by task instance.

6 RESULTS

Table 1 isolates composition-time control on the 38,482-candidate internal pool. Every arm traverses the same frozen candidate order within each task budget. *Checker accuracy* is the percentage of frozen candidates whose policy-specific replay emits a final output that passes the external task checker; an unrepaired rejection contributes zero after charging proposal and checking calls, while its unexecuted suffix is skipped. Appendix A.2.1 gives the candidate counts, conditional boundary-status denominators, and paired uncertainty.

Gate-only lifts checker accuracy from 72.3 to 82.7, accounting for 10.4 of the total 13.3 points; bounded repair contributes the remaining 2.9 points to reach 85.6. Domain counts and paired uncertainty in Appendix A.2.1 show the same ordering on Spider, UT-Python, and BBH-Logic.

Table 1: Internal-pool mechanism identification with GPT-4o-mini and a 10k-call ledger. Both rows preserve frozen candidate identity and order. Checker accuracy is candidate-level final-task correctness after the policy finishes; an unrepaired rejection scores zero. Admission supplies 10.4 of the 13.3-point fixed-pool gain; repair adds 2.9 points.

Isolation setting	Comparison	Headline number	Audit detail	Scope
Fixed-pool identification, gate only (38,482 frozen candidates)	Gate-only vs. the same no-filter pool	72.3 → 82.7 (+10.4)	Invalidity −73.6%; wasted-late budget −46.3%	—
Fixed-pool identification, gate + repair (38,482 frozen candidates)	Gate + bounded repair vs. the same no-filter pool	72.3 → 85.6 (+13.3)	Pre-repair CI 3.2%; CV among final-wrong 4.1%	—

Table 2 repeats the matched replay on a second frozen proposal pool and adds four timing/authority controls. Terminal validation changes final success by 0.9 points because detection follows the charged suffix. Eager node checks and a schema gate with full rerun recover more budget, yet SPLICEGUARD gate-only leads them by 6.6 and 4.5 points. Gate+repair reaches 78.1%, a 14.0-point lift over no filtering; BIRD improves from 62.8% to 76.4%, and ToolSandbox from 66.0% to 80.6%.

At the full operating point, the pre-repair boundary-invalid share among final-correct candidates is 3.6%, the certificate-valid share among final-wrong candidates is 4.4%, bounded repair recovers 68.7% of gate failures, and the mean suffix replay is 21.8% of a full execution. These conditional cells describe the policy's operating point; they are not a verifier-error guarantee. Together with the internal replay, the six-arm comparison shows that refinement authority and dependency-aware replay improve beyond generic early detection under the shared ledger.

Table 2: Second fixed-pool identification on 27,346 frozen candidates from BIRD dev and ToolSandbox. Task success and task-clustered 95% CIs use the instance denominator; remaining columns use the candidate denominator. Shared candidate order, cache policy, and 10k-call accounting isolate detection timing, refinement authority, repair, and replay locality.

Replay arm	Task succ. [95% CI]	Invalid (%)	Late d	Wasted late (%)	Verify (%)	API $
No filter	64.1 [63.3, 64.9]	28.9	0.79	37.6	0.0	6.88
Terminal validation	65.0 [64.2, 65.8]	27.6	0.74	34.9	1.6	6.91
Eager node-output validation	69.2 [68.4, 70.0]	18.4	0.53	25.7	3.2	6.72
Splice schema gate; full rerun	71.3 [70.5, 72.1]	15.2	0.44	21.8	3.7	6.64
SPLICEGUARD gate-only	75.8 [75.0, 76.6]	9.7	0.29	14.6	4.1	6.17
SPLICEGUARD gate+repair	78.1 [77.3, 78.9]	7.8	0.26	12.9	5.0	6.26

7 CONCLUSION

SPLICEGUARD establishes composition-time contracts as a throughput mechanism: refinement controls admission, deterministic repair restores eligible boundaries, and dependency-aware replay confines recomputation to the affected suffix. Matched freeze-then-replay identifies each contribution. Contract admission recovers 10.4 points and repair adds 2.9 on the 38,482-candidate internal pool; in the six-arm fixed-pool replay, gate-only reaches 75.8% and repair reaches 78.1%, compared with 69.2% for eager validation and 71.3% for a full-rerun schema gate. Coupling certificate authority to an explicit spend ledger makes strict-budget workflow composition auditable, testable under matched counterfactuals, and more efficient at converting avoided late work into usable inference budget.

REPRODUCIBILITY STATEMENT

The evaluation protocol, shared-budget accounting, environment and software versions, statistical procedures, fixed-pool policy-evaluation pseudocode, and formal proofs are specified in Sections 5–6 and Appendix A.1.1–A.4.

AI USE STATEMENT

Generative AI was used only for manuscript writing, editing, and presentation. The authors reviewed every AI-assisted edit and remain responsible for the final content.

REFERENCES

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. URL https://arxiv.org/abs/2110.14168v2.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 10764–10799, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tenenholtz. MRKL Systems: A

modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. arXiv preprint arXiv:2205.00445, 2022. URL https://arxiv.org/abs/2205.00445v1.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f1cf02ce09757f57c3b93c0db83181e0-Paper-Conference.pdf.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html.

Yu Li, Lehui Li, Zhihao Wu, Qingmin Liao, Jianye Hao, Kun Shao, Fengli Xu, and Yong Li. AgentSwift: Efficient LLM Agent Design via Value-guided Hierarchical Search. arXiv preprint arXiv:2506.06017, 2025. URL https://arxiv.org/abs/2506.06017.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A Stateful, Conversational, Interactive Evaluation Benchmark for LLM Tool Use Capabilities. arXiv preprint arXiv:2408.04682, 2024. URL https://arxiv.org/abs/2408.04682.

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. arXiv preprint arXiv:2303.17651, 2023. URL https://arxiv.org/abs/2303.17651v2.

Dheeraj Mekala, Jason Weston, Jack Lanchantin, Roberta Raileanu, Maria Lomeli, Jingbo Shang, and Jane Dwivedi-Yu. TOOLVERIFIER: Generalization to New Tools via Self-Verification. arXiv preprint arXiv:2402.14158, 2024. URL https://arxiv.org/abs/2402.14158v2.

George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pp. 106–119, 1997. doi: 10.1145/263699.263712. URL https://doi.org/10.1145/263699.263712.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9895–9901, 2021. doi: 10.18653/v1/2021.emnlp-main.779. URL https://aclanthology.org/2021.emnlp-main.779/.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv preprint arXiv:2303.11366, 2023. URL https://arxiv.org/abs/2303.11366.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them. arXiv preprint arXiv:2210.09261, 2022. URL https://arxiv.org/abs/2210.09261.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3911–3921, 2018. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425/.

Guibin Zhang, Kaijie Chen, Guancheng Wan, Heng Chang, Hong Cheng, Kun Wang, Shuyue Hu, and Lei Bai. EvoFlow: Evolving Diverse Agentic Workflows On The Fly. arXiv preprint arXiv:2502.07373, 2025. URL https://arxiv.org/abs/2502.07373.

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 *Proceedings of the 41st International Conference on Machine Learning*, pp. 62138–62160, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

A Supplementary Appendix

A.1 Execution Environment and Statistical Reporting

A.1.1 Environment and software

All controlled runs used 8×NVIDIA A100-80GB GPUs with CUDA 12.4, driver 550.54.14, Ubuntu 22.04.4, and Python 3.11.8. The shared source executor was GPT-4o-mini (2024-07-18) accessed through the OpenAI API. Task-specific tools included SQLite 3.45.3 for SQL evaluation, CPython 3.11 for unit tests, and jsonschema 4.22.0 for schema validation.

A.1.2 Statistical reporting

Where accuracy intervals are displayed, we report 95% confidence intervals over $n = 10$ independent seeds. Internal fixed-pool intervals use 1000 paired task-instance bootstrap resamples; the second fixed pool clusters resamples by task instance. Paired instance-level correctness comparisons use McNemar's test with Holm correction within each reported comparison family.

A.2 Fixed-Pool Decomposition and Six-Task Audit

A.2.1 Per-domain fixed-pool counts

Table A.1 decomposes the internal pool by domain. Each accuracy cell is the number of policy-specific final outputs accepted by the external checker divided by the original frozen-candidate denominator for that domain. The final column deliberately separates pre-repair boundary status from final outcome: "CI" is the share of final-correct candidates whose first boundary check was invalid, and "CV" is the share of final-wrong candidates whose first boundary check was valid. Thus a CI candidate can be repaired, admitted, and counted correct; an unrepaired rejection never contributes to a correctness numerator.

The paired McNemar discordant counts $(\text{first-only}, \text{second-only})$ are (438,2029)/(286,741) for Spider, (397,1722)/(264,633) for UT-Python, and (352,1437)/(231,519) for BBH-Logic, with pairs ordered as no-filter/gate-only then gate-only/gate+repair. At the final operating point, the checker-outcome $\times$ first-boundary-status cells $(CV, CI; WV, WI)$ are (11,552,394;97,2,161), (10,663,341;69,1,663), and (9,675,320;63,1,484). Here CV/CI denotes first-boundary certificate-valid/invalid among final-correct candidates and WV/WI the same split among final-wrong candidates.

A.2.2 Six-task protocol view

Table A.2 supplies the verifier-available task audit. Each row is a frozen-topology, task-instance-level accuracy across ten seeds; the split and per-seed task-instance denominator are printed in the first column. Accuracy cells show no-filter/gate-only/gate+repair with 95% CIs. Operational cells report gate+repair invalidity, no-filter→gate+repair failure lateness, and mean charged calls. These units differ from Table A.1, which reports candidate-level replay accuracy.

Table A.1: Per-domain decomposition of the 38,482-candidate replay. Each accuracy cell reports correct final outputs / original frozen candidates, percentage, and paired task-instance bootstrap 95% CI. The last column reports pre-repair CI among final-correct candidates / pre-repair CV among final-wrong candidates at the gate+repair operating point; these are conditional boundary-status shares, not false-reject or false-accept rates.

Domain	No filter	Gate-only	Gate+repair	Pre-repair CI / CV
Spider	9,900/14,204; 69.7 [68.8,70.6]	11,491/14,204; 80.9 [80.1,81.7]	11,946/14,204; 84.1 [83.4,84.8]	3.3/4.3
UT-Python	9,310/12,736; 73.1 [72.2,74.0]	10,635/12,736; 83.5 [82.7,84.3]	11,004/12,736; 86.4 [85.7,87.1]	3.1/4.0
BBH-Logic	8,622/11,542; 74.7 [73.8,75.6]	9,707/11,542; 84.1 [83.3,84.9]	9,995/11,542; 86.6 [85.9,87.3]	3.2/4.1

Table A.2: Six-task audit under the shared GPT-4o-mini executor and 10k-call ledger. All rows use ten seeds and frozen test topologies; the first column states the split and per-seed task-instance denominator.

Task / split	No filter [CI]	Gate [CI]	Repair [CI]	Operational audit
BBH-Logic / test / 250	89.4 [88.0,90.8]	91.2 [89.9,92.5]	96.9 [95.9,97.9]	Inv. 4.8; late 0.62→0.24; calls 9,340
Boolean Expr. / test / 250	91.8 [90.5,93.1]	93.0 [91.8,94.2]	97.8 [96.9,98.7]	Inv. 3.9; late 0.59→0.21; calls 9,210
Dyck / test / 250	86.7 [85.1,88.3]	89.9 [88.4,91.4]	94.8 [93.6,96.0]	Inv. 6.4; late 0.66→0.25; calls 9,460
UT-Python / test / 164	66.4 [62.1,70.7]	68.7 [64.5,72.9]	74.2 [70.3,78.1]	Inv. 8.9; late 0.71→0.28; calls 9,760
GSM-Hard / test / 250	78.5 [76.6,80.4]	81.4 [79.6,83.2]	86.1 [84.5,87.7]	Inv. 5.7; late 0.64→0.23; calls 9,520
Spider / dev / 1,034	77.1 [75.3,78.9]	78.9 [77.2,80.6]	84.6 [83.1,86.1]	Inv. 7.6; late 0.69→0.26; calls 9,880

A.3 FIXED-POOL POLICY-EVALUATION LOOP

Algorithm 1 specifies the policy-evaluation semantics used by the controlled replays. It defines charging and final-outcome handling for a frozen, ordered proposal stream; it does not specify a search objective, archive replacement rule, or candidate tie-breaker.

Algorithm 1 SPLICEGUARD fixed-pool policy evaluation

Input: frozen ordered candidate stream $\mathcal{P}$, call budget B, repair limit $K=3$, certificate verifier $\mathrm{CertCheck}(\cdot)$, external checker Check
Init: ledger $\mathcal{L} \leftarrow \emptyset$; remaining budget $b \leftarrow B$
for all c in $\mathcal{P}$ in its frozen order **do**
  if the next required proposal or check charge exceeds b **then**
    break
  end if
  charge the proposal and first boundary check; append their outcomes to $\mathcal{L}$
  test refinement and the applicable boundary predicates
  if the boundary fails and no registered repair is eligible **then**
    record a charged rejection with final correctness zero; continue
  end if
  if the boundary fails and repair is eligible **then**
    for $k = 1$ to K **do**
      charge one deterministic repair and re-check the affected boundary
      if the repaired boundary passes **then**
        break
      end if
    end for
    if the boundary still fails **then**
      record a charged rejection with final correctness zero; continue
    end if
  end if
  execute the admitted downstream dependency closure with shared caching
  charge the final check, assign $\mathsf{Check}(c)$, and append the final outcome to $\mathcal{L}$
end for
Return: $\mathcal{L}$

A.4 PROOFS

We restate the contract-refinement definition and the splice-boundary soundness proposition summarized in Section 4. Downstream replay and certificate re-checking extend this boundary guarantee through the affected dependency closure, while the external checker evaluates the final task output.

Definition 1 (Contract refinement)**.** *An interface* $\mathcal{I}_2 = (\mathcal{T}_{in}^{(2)}, \mathcal{T}_{out}^{(2)}, \Pi_2, \mathcal{E}_2^\star)$ refines $\mathcal{I}_1 = (\mathcal{T}_{in}^{(1)}, \mathcal{T}_{out}^{(1)}, \Pi_1, \mathcal{E}_1^\star)$*, written* $\mathcal{I}_1 \preceq \mathcal{I}_2$*, if (i)* $\mathcal{T}_{in}^{(2)} \supseteq \mathcal{T}_{in}^{(1)}$ *and* $\mathcal{T}_{out}^{(2)} \subseteq \mathcal{T}_{out}^{(1)}$*, (ii)* $\Pi_1 \subseteq \Pi_2$*, and (iii)* $\mathcal{E}_1^\star \subseteq \mathcal{E}_2^\star$*.*

Proposition 2 (Compositional soundness at a splice boundary)**.** *If a module with interface* $\mathcal{I}_2$ *replaces a module required to satisfy* $\mathcal{I}_1$ *at a splice boundary, the replacing module actually satisfies the types, predicates, and evidence obligations of its certified interface on this input, and* $\mathcal{I}_1 \preceq \mathcal{I}_2$*, then the splice boundary remains contract-sound: every required type, required predicate, and required evidence field of* $\mathcal{I}_1$ *is satisfied after the splice.*

Proof. Let x be an upstream artifact valid for $\mathcal{I}_1$, so $x \in \mathcal{T}_{in}^{(1)}$. By Definition 1, $\mathcal{T}_{in}^{(2)} \supseteq \mathcal{T}_{in}^{(1)}$, so the replacing module accepts x. Let y be the produced artifact. By the satisfaction premise and refinement, $y \in \mathcal{T}_{out}^{(2)} \subseteq \mathcal{T}_{out}^{(1)}$, so the output type required by $\mathcal{I}_1$ is preserved. Because $\Pi_1 \subseteq \Pi_2$ and $\mathcal{E}_1^\star \subseteq \mathcal{E}_2^\star$, every predicate and evidence field required by $\mathcal{I}_1$ is satisfied by the verified replacement certificate. Interface declarations alone do not establish this premise. Hence the splice boundary satisfies every requirement of $\mathcal{I}_1$. □

A.4.1 PROOF OF PROPOSITION 1

Assume $C > 0$, $0 < \pi \leq 1$, $0 \leq \rho' \leq \rho < 1$, and $d, d' \in [0, 1]$, so the denominators below are positive. Write $A = 1 - \rho$, $B = \rho d$, $A' = 1 - \rho'$, $B' = \rho' d'$, $R = \rho' r$, and $Q = \rho' rq$. Then

$$\mathcal{T}_0 = \frac{\pi A}{C(A + B)}, \qquad \mathcal{T} = \frac{\pi(A' + R)}{C(o + A' + B' + Q)}.$$

Cross-multiplying the positive denominators yields the exact dominance condition

$$o \;\leq\; \frac{A'B + R(A + B)}{A} - B' - Q.$$

Because $\rho' \leq \rho$, we have $A' \geq A$ and therefore $A'B/A \geq B$; because $A + B \geq A$, we also have $R(A + B)/A \geq R$. Hence a sufficient condition is

$$o \;<\; B - B' + R - Q.$$

Finally, since $d' \leq 1$, we have $R \geq Rd'$, which gives the main-text condition

$$o \;<\; B - B' + Rd' - Q \;=\; \rho d - \rho'(1 - r)d' - \rho' rq.$$

This bound is conservative but sufficient, and it becomes strict whenever $\rho > \rho'$ or $d' < d$.