StateScope: Interactive, Grounded Diagnosis and Repair of Solution-State Drift in Mathematical Reasoning

L D M S Sai Teja[1,2], **Sangam Sai Anish**[1], **and Akshaj Agadi**[1,3]
[1]Exea Labs [2]NIT Silchar, India [3]Texas A&M University, USA
saitejalekkala05@gmail.com

Project: https://statescope.github.io/StateScope/ Video: https://youtu.be/YYa2EfuAiHk

Abstract

Final-answer accuracy hides where a mathematical solution first becomes inconsistent and whether a later correction merely masks that error. We present StateScope, an interactive system for detecting, localizing, and repairing *solution-state drift*: divergence between a solver's evolving state and a symbolically verified reference trajectory. Under a controlled comparison, two solvers receive the same Computer Algebra System (CAS)-derived operation schedule and typed arguments. We provide two systems, 1) System C records the model's textual state claim, 2) System D rederives state in an external typed ledger. A SymPy oracle checks every transition and explains component-level differences without an LLM judge. Users can step through a run, edit an operation at any retained step, directly repair System C's claimed state, continue the model, and repeatedly branch from earlier interventions. Our paired evaluation covers 250 problems and twelve open and proprietary models, and a human study measures diagnostic accuracy and time. StateScope targets reasoning researchers, model developers, and educators who need to inspect and manipulate more than a final answer.

1 Introduction

A model can report the right mathematical answer while carrying a wrong binding, dropped constraint, or stale intermediate expression. Conversely, a valid operation can be applied to an already corrupted state. An outcome score collapses these cases into one bit and cannot answer three questions needed for debugging: ***where did the state first diverge***, ***what changed***, and ***could a targeted repair recover the run?*** Process supervision and error-location benchmarks make intermediate correctness visible (Lightman et al., 2024; Zheng et al., 2025), but usually score fixed text rather than an executable state that users can revise. We call the first divergence between the model-carried solution state and the symbolically verified reference state at an aligned step the *onset of solution-state drift*.

StateScope makes mathematical reasoning an inspectable sequence of typed state updates. It combines dynamic problem generation, a one-operation-per-turn model protocol, an external computer-algebra-system (CAS) oracle, and an interactive branching interface. Its flagship experiment compares two state-ownership policies while holding the operation schedule fixed: System C trusts the model's complete textual state claim; System D treats typed ledger state as authoritative. The comparison isolates bookkeeping from operation selection, while a separate autonomous mode remains available for open-ended diagnosis. Researchers can test whether gains come from better planning or safer state management. Developers can replay failures in tool-using mathematical agents. Educators can show students how one local error propagates and how a correction changes the remainder of a derivation. The system does not claim to certify arbitrary proofs; it provides an auditable, CAS-grounded laboratory for mathematical state.

Our contributions: (1) a formalization and component-level measurement of solution-state drift; (2) a controlled comparison that isolates model-owned from ledger-owned state under the same verified operation schedule; (3) repeatable, multi-step interventions that support model continuation, deterministic re-derivation, and persistent branch histories; and (4) an extensible demonstration runtime with dynamic instances, twelve-model evaluation support, and auditable exports.

2 System Design

2.1 Dynamic problems and verified traces

Each problem family samples fresh parameters from a seeded template and constructs a reference

Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 82–88
November 6-10, 2026 ©2026 Association for Computational Linguistics

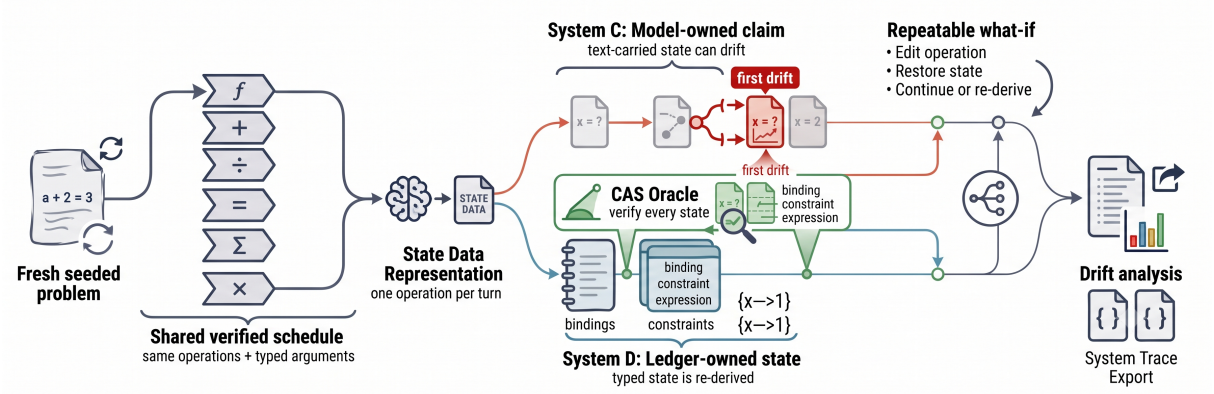


Figure 1: StateScope pipeline. System-C: model-owned claims, where text-carried states are susceptible to drift; System-D: ledger-owned states, where typed states are continuously verified by a CAS Oracle.

trace $G = (G_0, \ldots, G_{T-1})$ with SymPy (Meurer et al., 2017). The current demonstration covers binding chains, equations with domain or branch constraints, coupled recurrences, and derivative lemma graphs. A manifest records the seed, family, generator version, source policy, and code revision. Regeneration therefore changes surface instances while remaining reproducible. This reduces direct instance memorization. At step t, the structured state S_t contains live bindings, constraints, a current expression or equation, candidate solutions, dependency nodes, and an optional final answer. Symbolic expressions are normalized and compared semantically. The oracle trace is constructed independently of the model response.

2.2 Typed operation protocol

At each turn, the model receives the problem, current state, allowed operation schemas, and progress guardrails. In controlled mode, the operation and typed arguments are additionally prescribed by the verified reference schedule; in autonomous mode, the model selects them. The model emits a JSON object containing the operation, typed arguments, a complete claimed post-state, a completion flag, and a short rationale.

2.3 Two state-ownership policies

System C: model-owned claimed state. The requested operation is executed on a scratch ledger, but the model's textual post-state becomes the state passed to the next turn. A missing constraint or stale binding can therefore propagate even if the current operation is valid.

System D: ledger-owned typed state. The runtime validates the operation, applies it transactionally, and re-derives the post-state from the typed ledger. Direct state edits are rejected by design. An invalid or unverifiable operation leaves the last safe state intact. System D is not assumed to improve planning; it tests whether external state ownership prevents state-carrying errors.

In the recommended *controlled* comparison, both systems receive the same CAS-derived operation and arguments at every aligned step. The model still emits a claimed state, which C carries and D ignores. An *autonomous* diagnostic mode instead asks the model to choose each operation and separately labels schedule divergence. A premature answer in controlled mode is recorded but does not truncate the fixed schedule. Autonomous mode is provided as an exploratory diagnostic feature; all quantitative results reported in this paper use the controlled mode unless explicitly stated otherwise.

2.4 Drift localization and metrics

StateScope compares S_t and G_t component by component. The corruption-onset depth (COD) is the first unequal aligned step; propagation length is the number of later unequal steps. The interface distinguishes a CAS-invalid operation from a CAS-valid operation applied to a stale state and renders deterministic explanations such as "constraint $x \neq 3$ is absent." Because a CAS-valid alternative path can differ from the canonical trace, *drift* means reference-trajectory divergence, not automatically a proof of invalid mathematics. We report final-answer accuracy (FA), and strict gold-normalized state fidelity SF_g, which assigns zero credit to omitted steps. For System C, state-repair Recovery@1 (C_{state} R@1) measures whether replacing the first drifted claimed state with the oracle state and continuing the same model yields the cor-

rect final answer. For System D, operation-repair Recovery@1 (D_{op} R@1) replaces the first failed operation with the oracle operation, re-derives the ledger, and then continues execution. Both metrics are conditional on runs containing an eligible intervention event and are reported separately because they represent different repair mechanisms. For comparisons across traces of different lengths, we additionally report normalized drift onset as the first-drift position divided by the number of aligned steps in the verified trajectory.

$$\mathrm{SF}_g = \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{1}[t \text{ emitted and } S_t \equiv G_t], \quad (1)$$

2.5 Interactive, repeatable intervention

The browser offers *Run both* for a complete paired trace and *Step-through* for one operation at a time. Live reasoning indicators show which panel is awaiting a response. Every completed, failed, or omitted step remains selectable. At any retained step, a user may edit the operation for either system or directly edit System C's claimed state. A one-click control copies the oracle state. The user can then continue with the same model or re-derive downstream deterministically.

Each intervention creates a child branch without overwriting its parent. The new branch can itself be edited, and the interface retains up to 128 branches per system in one run. System D supports repeated operation edits but never direct state edits. JSON and Markdown exports include model and route provenance, operations, states, verdicts, failure events, metrics, and branch ancestry. This makes the framework a replayable experiment rather than a one-shot.

3 Evaluation

3.1 Research questions and protocol

We ask: **RQ1**, *Does ledger ownership improve final accuracy and state fidelity under a fixed operation schedule?* **RQ2**, *Where does model-owned state first drift and how far does it propagate?* **RQ3**, *Can a targeted intervention at the first eligible error recover a run?* **RQ4**, *Does the interface help people identify drift more accurately and quickly than raw traces?*

DemoBench contains 250 generated problems balanced 63/63/62/62 across the four families above. Every model-problem pair runs once in C and D under controlled comparison: 12 models $\times$ 250 problems $\times$ two systems = 6,000 primary runs. Temperature is zero where supported; reasoning effort and token budgets are fixed by provider class. We store exact snapshot IDs, prompts, package versions, seeds, latency, token counts, and errors. The six open models are Qwen2.5-1.5B-Instruct, Qwen3-4B, Qwen3-8B, Qwen3-30B-A3B-Instruct-2507, Gemma-3-12B-IT, and Mistral-Small-3.1-24B-Instruct-2503 (Qwen et al., 2025; Yang et al., 2025; Team et al., 2025; Mistral AI, 2025). The six proprietary models are GPT-5.1, GPT-5.4 mini, GPT-5.4 nano, GPT-5.6 Luna, Gemini 2.5 Flash, and Gemini 2.5 Pro. Open models are served locally through vLLM (Kwon et al., 2023); proprietary models use native APIs through the same typed adapter.

We use paired bootstrap confidence intervals over problems and paired randomization tests for ΔFA and $\Delta\mathrm{SF}_g$, with Holm correction across primary comparisons. Family-level results, failure counts, coverage, latency, and cost are reported in the artifact; no failed API call is converted into an incorrect math answer without a separate failure label.

3.2 Quantitative Analysis

Table 2 shows that the controlled schedule makes ledger execution highly reliable while leaving C exposed to state-carrying errors. The macro-average improvement from C to D is 37.6 points for final-answer accuracy (95% CI: [35.9, 39.4]) and 43.4 points for strict state fidelity (95% CI: [41.8, 45.0]). Paired randomization tests yield $p < .0001$ for both comparisons and remain significant after Holm correction ($p_{\text{Holm}} < .0002$).

Table 1 reports the model-level results of the controlled comparison. Ledger ownership improves both final-answer accuracy and strict state fidelity for every evaluated model. Even the highest System C final-answer accuracy, 78.4%, remains below the lowest System D accuracy, 90.8%. The same separation appears in state fidelity: System C ranges from 18.6% to 73.1%, whereas System D ranges from 92.5% to 99.2%. This consistency across model sizes and access types indicates that the observed effect is not attributable to a single provider or model family. The two Recovery@1 columns measure different interventions and should therefore not be interpreted as a direct system ranking. State repair recovers 71.4% of eligible System C runs on average, showing

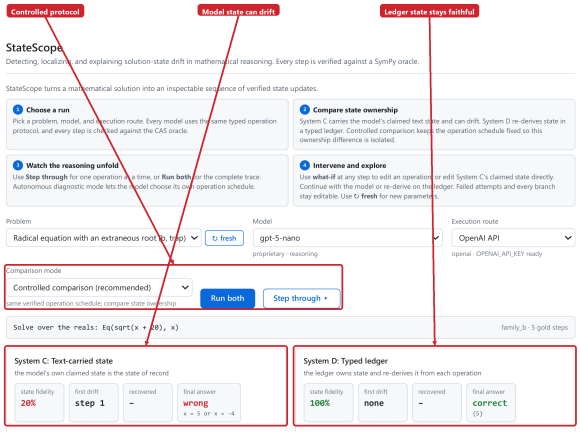


Figure 2: Component-level localization of the first drift during the demonstration.

Access	Model	FA (%)		SF_g (%)		Recovery@1 (%)	
		C	D	C	D	C_{state}	D_{op}
Open	Qwen2.5-1.5B	22.4	90.8	18.6	92.5	47.9	78.6
	Qwen3-4B	34.8	93.2	30.7	94.6	55.1	83.2
	Qwen3-8B	46.0	95.2	41.8	96.4	62.7	87.5
	Qwen3-30B-A3B	61.2	97.2	56.4	98.0	71.5	92.1
	Gemma-3-12B	48.4	95.6	43.7	96.8	64.9	88.4
	Mistral-Small-3.1-24B	57.6	96.8	52.2	97.6	69.8	90.7
Closed	GPT-5.1	75.2	98.4	70.3	99.0	82.6	95.6
	GPT-5.4 mini	72.8	98.0	67.5	98.8	80.9	94.8
	GPT-5.4 nano	63.6	97.2	58.4	98.0	74.3	92.9
	GPT-5.6 Luna	78.4	98.8	73.1	99.2	85.4	96.7
	Gemini 2.5 Flash	68.8	97.6	63.2	98.4	77.5	93.8
	Gemini 2.5 Pro	76.4	98.4	71.6	99.0	83.8	95.9
Macro average		58.8	96.4	54.0	97.4	71.4	90.9

Table 1: Paired controlled-comparison results. C carries model-claimed state, whereas D re-derives state in the ledger. C_{state} R@1 restores C's first drifted claimed state to the oracle state. D_{op} R@1 replaces D's first failed operation with the oracle operation and re-derives the ledger.

that correcting the first corrupted state frequently changes the eventual outcome. Operation repair recovers 90.9% of eligible System D runs, indicating that preserving a safe ledger state makes failures more locally repairable. Neither result implies that one intervention guarantees recovery, because the continued model execution may introduce another error.

Drift onset and propagation. Among System C runs exhibiting state drift, the first divergence occurs at a median normalized depth of 0.44 of the verified trajectory, corresponding to approximately the third aligned step. Following the first divergence, the state remains mismatched for a median of 3 subsequent aligned steps, and 67% of drifted runs remain divergent at the final aligned state. These results address RQ2 and show that model-owned state errors are often persistent rather than isolated: once an incorrect binding, constraint, or expression enters the carried state, it can affect subsequent otherwise valid operations.

Primary comparison	**Effect**	**95% CI**	**Randomization p**	**Holm-adjusted p**
ΔFA, D minus C	+37.6	[35.9, 39.4]	$< .0001$	$< .0002$
ΔSF_g, D minus C	+43.4	[41.8, 45.0]	$< .0001$	$< .0002$
C state-repair R@1	71.4%	[68.9, 73.8]	—	—
D operation-repair R@1	90.9%	[87.6, 93.4]	—	—

Table 2: 95% confidence intervals, and significance testing for the primary comparisons.

Because C and D receive identical operations and typed arguments, the paired differences isolate state ownership within the controlled regime. They do not measure autonomous planning ability. The confidence intervals and randomization tests therefore support the narrower conclusion that external ledger ownership prevents errors caused by carrying an incorrect textual state through an otherwise fixed operation schedule.

3.3 Human evaluation

A counterbalanced study with 18 participants familiar with undergraduate algebra and calculus. Each participant diagnoses 12 traces, split evenly



Figure 3: A repair that remains available for further interventions.

Measure	Raw trace	StateScope
First-drift accuracy (%)	58.3	88.9
Median time (seconds)	72.4	25.8
Explanation correct (%)	61.1	91.7
Usefulness (1-5)	2.7	4.5

Table 3: Human evaluation results.

Access	Coverage		Failures		Median latency	
	C	D	C	D	C	D
Open models	96.1%	97.2%	58/1,500	42/1,500	14.1 s	13.5 s
Closed models	98.5%	98.8%	23/1,500	18/1,500	24.6 s	23.2 s
Overall	97.3%	98.0%	81/3,000	60/3,000	19.4 s	18.4 s

Table 4: Coverage, failure rates, and median latencies grouped by model access type.

between a raw JSON/Markdown trace and the StateScope interface; problem, condition order, and drift family are counterbalanced. Participants mark the first drift step and component and briefly explain the error. Primary outcomes are first-drift accuracy and completion time; secondary outcomes are explanation correctness and five-point usefulness. As shown in Table 3, the interface improves first-drift accuracy by 30.6 percentage points and explanation correctness by 30.6 points relative to raw traces. Median diagnosis time decreases by 46.6 seconds, corresponding to a 64.4% reduction, while mean usefulness increases by 1.8 points on the five-point scale. These results address RQ4 by showing that component-level state differences and navigable execution histories make drift both easier and faster to diagnose.

4 Coverage and Latency

Table 4 shows that the reported mathematical metrics are supported by high execution coverage. System D completes 98.0% of its 3,000 runs, compared with 97.3% for System C, and records 21 fewer failures overall. Its median latency is also one second lower. Separating these failures from mathematical errors prevents provider outages, malformed responses, and tool failures from being counted as incorrect solutions.

5 Comparison with Existing Systems

Outcome verifiers score completed solutions (Cobbe et al., 2021); process supervision and ProcessBench assess intermediate text or locate an incorrect step (Lightman et al., 2024; Zheng et al., 2025). ReasonGraph visualizes sequential and branching reasoning paths (Li et al., 2025). Formal proof assistants expose machine-checked proof states but require a formal language. StateScope occupies a complementary middle ground: natural-language models act through a small typed operation language, a CAS checks supported transitions, and users edit and replay a live stateful computation. Our framework's technical novelty is therefore not any single component in isolation. It is the causal comparison of state ownership under an identical verified schedule, coupled with online component diffs and interventions that can be repeated on their own descendants. This design turns drift from a post-hoc label into an editable experimental object. Table 5 summarizes this positioning. Unlike outcome verifiers and fixed process benchmarks, StateScope exposes a live typed state and permits repeated counterfactual interventions. Its scope is narrower than that of a proof assistant because CAS verification is limited to the supported mathematical families, but it requires neither a complete formal proof language nor a manually authored proof script.

6 Audience, Availability, and License

The primary audience is researchers studying reasoning, process supervision, and tool-using agents. Model developers can use exported traces to reproduce parser, planner, and state-management failures. Educators and demonstrators can use fresh instances and the step-

System type	Live run	Typed state	CAS check	Repeated what-if
Outcome verifier	–	–	–	–
Process benchmark	–	–	–	–
Reasoning visualizer	yes	–	–	limited
Proof assistant UI	yes	yes	proof	yes
StateScope	yes	yes	yes	yes

Table 5: Capability-level positioning. StateScope emphasizes CAS-grounded state comparison and repeatable intervention, not full formal proof certification.

through view to teach error propagation. The Python backend, dependency-free browser client, generation scripts, model configurations, scorer, and offline tests are at https://github.com/saitejalekkala33/DriftMath. The installable or live demonstration will be at https://statescope.github.io/StateScope/, and the video at https://youtu.be/YYa2EfuAiHk.

7 Conclusion

StateScope addresses a gap between final-answer scoring and full formal proof: it exposes the evolving state of a mathematical solver, checks supported transitions against a CAS oracle, and lets users repair and replay the point where state first drifts. The controlled C/D comparison identifies the effect of state ownership; the branchable interface shows whether a correction actually survives subsequent reasoning. Dynamic generation, provider-independent typed operations, and auditable exports make the same demonstration useful for research, development, and teaching.

Limitations and Responsible Use

The oracle covers the operation schemas and mathematics encoded by the current families; it does not verify arbitrary natural-language proofs. SymPy can reject informal or ambiguous notation, and canonical step alignment can flag a valid alternative path as drift. We therefore display parser and CAS uncertainty and do not equate every highlighted difference with mathematical invalidity. Controlled mode isolates state ownership but is easier than autonomous planning; both views must be reported separately. Fresh parameters reduce direct item memorization but do not remove template familiarity. Single deterministic samples do not measure decoding variance, and proprietary aliases can change unless snapshot identifiers are archived. The system should not certify safety-critical calculations. Human-study results may depend on participants' mathematical background.

Acknowledgments

We express our sincere gratitude to Exea Labs[1] for providing compute access to their AMD Instinct MI300X (192GB) GPU infrastructure. All open-weight models evaluated in this work were executed on this platform, which significantly accelerated our experiments and made large-scale evaluation feasible.

References

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

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Zongqian Li, Ehsan Shareghi, and Nigel Collier. 2025. Reasongraph: Visualization of reasoning methods and extended inference paths. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 140–147.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601.

Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. 2017. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103.

Mistral AI. 2025. Mistral-Small-3.1-24B-Instruct-2503 model card. https://huggingface.co/mistralai/Mistral-Small-3.1-24B-Instruct-2503. Hugging Face model card.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

[1] https://www.exealabs.org/

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. Processbench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1009–1024.

A Demonstration Script

The video uses one coupled-recurrence instance. First, we select the problem, exposes the model menu, chooses GPT-5.4 nano, and starts controlled *Run both*. Second, the paired panels reveal the first C drift while D remains faithful to the ledger. Third, the presenter opens *what-if* at that step, restores the oracle state, and continues with the model. If C drifts again, the intervention is repeated to demonstrate that branches are not one-shot. Finally, the presenter reaches the correct answer, revisits a prior branch, and exports the run. On-screen labels identify the shared schedule, CAS verdict, first drift, branch ancestry, and the distinction between C state edits and D operation edits.

B Failure breakdown

Table 6 further separates the failure sources. Unrepaired JSON or protocol errors are the largest category for both systems, accounting for 42 failures in C and 31 in D. CAS or tool-application failures decrease from 21 to 12 under ledger ownership, while provider and runtime failures remain similar. The category totals exactly reproduce the 81 C failures and 60 D failures reported in Table 4.

Failure type	**System C**	**System D**
Provider or runtime failure	18	17
Unrepaired JSON/protocol failure	42	31
CAS/tool-application failure	21	12
Total	81	60

Table 6: Observed failure breakdown for the controlled-comparison runs.

C API costs

Table 7 reports a total proprietary-model API expenditure of $66.50. GPT-5.1 and Gemini 2.5 Pro together account for $37.30, or 56.1% of this total. These values represent provider API expenditure and do not estimate the hardware, electricity, or engineering costs of locally serving the open models.

The released artifact records token counts and the pricing schedule used for reproducibility.

Model	**API cost**
GPT-5.1	$18.80
GPT-5.4 mini	$9.50
GPT-5.4 nano	$2.70
GPT-5.6 Luna	$12.30
Gemini 2.5 Flash	$4.70
Gemini 2.5 Pro	$18.50
Total closed-model cost	**$66.50**

Table 7: Breakdown of cumulative API costs associated with evaluating the closed models.