Failure-Learning Claims Need Controls: A Claim-Control Evaluation of Language Agents

Anonymous submission

Abstract

Failure-learning papers often ask whether an agent improves after a failed trace, verifier message, or self-reflection. The scientific question is sharper: which claim does that improvement support? A single repair or transfer score cannot tell whether the agent repaired the same instance, transferred a semantic invariant, solved a nearby target directly, copied a source rule where it was invalid, or merely failed an over-specific gold plan. We introduce a claim-control evaluation for failure-learning agents that reports these events separately. Across 41,178 controlled DeepSeek rows, the same outputs support opposite conclusions under exact and core-order scoring: with non-leaky constraint feedback, positive core transfer reaches 0.982–1.000 while exact success remains 0.023–0.187, and contrastive core target-solving reaches 0.939–0.999 while exact success remains 0.000–0.018. We validate this boundary with a 92,160-row executable tool-use run, where the maximum core-minus-exact gap is 0.628 and 1,417 core-valid rows fail execution. A balanced 11,520-row audit across four configurations from three model families replicates the strict–core gap (0.393–0.591); core–execution gaps are smaller and model-dependent (0.000–0.031). Task-cluster bootstrap intervals resample eight source tasks rather than treating rows as independent. Only one row exhibits invalid source-rule transfer, so the evidence rejects a broad claim that agents generally overtransfer failures. It supports the narrower methodological claim that failure-learning conclusions are unstable unless evaluations separately report repair, transfer, target-solving, invalid transfer, scoring granularity, executable validity, invariant baselines, and run accounting. **Content areas:** ML, MAS, NLP.

Introduction

When a language agent fails, modern systems rarely discard the episode. They store traces, summarize verifier feedback, reflect on the error, retry with a modified plan, or extract a rule for future use (Shinn et al. 2023; Zhao et al. 2024; Madaan et al. 2023; Wang et al. 2023). These mechanisms are often described as learning from failure. The phrase is useful but scientifically underspecified. A later success may mean that the agent replayed a local correction, extracted a transferable invariant, solved a new target without using the source rule, or simply matched a relaxed metric that hides an unsafe extra operation.

The central question is therefore not whether a score improved after a failure. It is which scientific claim that improvement supports. A memory method, a reflection prompt, and a direct invariant prompt can all improve a later answer, but those improvements imply different mechanisms. Likewise, a contrastive target can be solved because the agent understood the target, not because it withheld a source rule. Failure learning needs a reporting standard that keeps these interpretations separate.

This ambiguity changes the conclusion of an experiment. If a benchmark reports only exact-plan success, a semantically valid repair can be marked wrong because it omits an auxiliary cleanup action. If it reports only relaxed success, a hallucinated operation can be hidden behind the correct dependency order. If it reports only contrastive success, target-solving can be mistaken for controlled non-transfer. If it reports only aggregate improvement, a memory or reflection method may receive credit for a gain that a direct invariant prompt already explains.

We propose the claim-control surface as a reporting standard, not as another single benchmark score. A failure-learning evaluation should ask whether the agent repaired the source, transferred the core invariant to a positive target, reproduced the full gold plan, omitted auxiliary hidden actions, solved a contrastive target on its own terms, invalidly copied the source rule into a target where that rule was wrong, and produced an executable output. These are different scientific claims. Compressing them into one score is where the pathology enters.

The project began with a stronger negative hypothesis: agents would repair locally but fail to transfer semantic counterexamples. That claim did not survive calibration. Once the task text made the dependency boundary clear, DeepSeek often transferred the core invariant and solved contrastive targets well. The surviving contribution is therefore methodological and, in our view, more useful: many failure-learning conclusions are unstable under missing controls. The right lesson is not that agents cannot learn from failure, but that existing evaluations can make the same outputs look like failure or success depending on which claim they silently score.

The resulting story is a claim flip, not a leaderboard. Exact-only reporting makes the agents look as if they mostly fail to learn from failure. Core-order reporting makes the same outputs look near-ceiling after calibration. Executable validation then shows why neither metric is sufficient alone. Our contributions are:

- We propose a claim-control reporting standard for failure-learning agents that separates local repair, exact positive transfer, core-order positive transfer, hidden-action miss, contrastive target-solving, invalid source-rule transfer, and executable validity.
- We show a large strict-versus-core reversal in 41,178 controlled DeepSeek rows: calibrated core transfer is near ceiling while exact-plan success remains near zero for many groups.
- We identify invariant prompting as a necessary baseline. It is not a new method, but it is strong enough that gains from memory or reflection cannot be interpreted without it.
- We validate the scoring boundary with a 92,160-row executable workflow adapter and a balanced 11,520-row audit of four configurations spanning DeepSeek, Qwen, and Mistral families, with uncertainty clustered by source task.
- We document a run-accounting failure mode: asynchronous API evaluations can write misleading completion markers unless valid rows and worker exits are checked directly.

Claim-Control Evaluation

Task structure. Each example contains a source failed plan, a source repair, a target failed plan, a target repair, a set of allowed operations, and optional middle actions. The positive split maps a source failure to a lexically distinct target with the same invariant. The contrastive split maps it to a near-neighbor target with a different invariant. For example, a source rule about reserving inventory before charging a customer may be paired with a positive target that uses different operation names but the same dependency, or with a contrastive target where the correct rule is to snapshot state before deletion.

Metrics. Let p be the predicted target plan, g the gold plan, and O the set of optional actions. Exact success requires ordered equality,

$$S_{\text{exact}}(p, g) = \mathbb{1}[p = g].$$

Core-order success removes optional actions from the gold plan and asks whether the remaining required subsequence appears in the prediction:

$$S_{\text{core}}(p, g, O) = \mathbb{1}[c(g, O) \sqsubseteq p].$$

Hidden-action miss records whether optional gold operations are omitted. On contrastive examples, invalid source-rule transfer is counted only when the prediction copies or prefixes the source repair in a target where that source repair is invalid. This is deliberately separate from contrastive target-solving. A model can solve a contrastive target correctly without transferring the source rule, and that should not be scored as overgeneralizing.

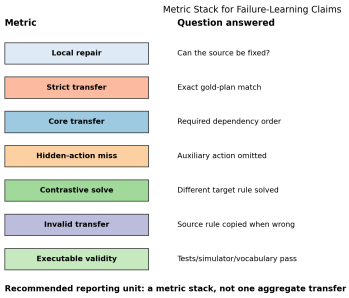

Figure 1: Claim-control stack. Each metric answers a different scientific question, so replacing the stack with a single aggregate transfer score loses claim-critical information.

Control	Claim it supports	Risk if omitted
Local repair	The source failure is fixable.	Transfer is interpreted before repair is established.
Exact success	The full gold plan is reproduced.	Exact reproduction is confused with semantic validity.
Core order	The required dependency is preserved.	Valid repairs become false negatives.
Hidden miss	Auxiliary actions are omitted.	Relaxed scores hide underspecification.
Contrastive target	The nearby target is solved.	Target-solving is confused with non-transfer.
Invalid transfer	The source rule is copied where wrong.	Overgeneralization is inferred from target failure.
Executable validity	The output is behaviorally safe.	Core-valid but unsafe plans are over-credited.

Table 1: Why the evaluation reports a stack rather than one transfer score.

Baselines and feedback. We evaluate raw trace, Reflexion-style verbal reflection, semantic grammar, and invariant rule prompting. The invariant baseline asks directly for the scoped dependency that should transfer. It is included to test whether apparent memory or reflection gains are actually explained by invariant abstraction. We compare no-feedback transfer with a non-leaky constraint-feedback condition that states a dependency boundary, such as one operation needing to precede another, but does not reveal the full ordered target plan.

Scale and accounting. The controlled scale-up contains 41,178 valid rows, 82,371 API calls, and 55.66M tokens. The intended run was larger, but the DeepSeek account hit a 402 insufficient-balance error after about 28–29 effective chunks. The original launcher could still write later completion markers after failed background workers. We therefore count evidence only from valid JSONL rows. The real-LLM executable workflow run completes 92,160 rows, 92,161 API

Mode	Baseline	Pos-E	Pos-C	Ctr-E	Ctr-C	Inv.
None	Invariant	.109	.492	.018	.379	.000
None	Raw trace	.111	.674	.016	.468	.007
None	Reflexion	.006	.295	.000	.360	.000
None	Sem. grammar	.142	.626	.001	.455	.000
Constraints	Invariant	.059	.982	.001	.960	.000
Constraints	Raw trace	.060	.998	.018	.968	.000
Constraints	Reflexion	.023	1.000	.000	.939	.000
Constraints	Sem. grammar	.187	1.000	.001	.999	.000

Table 2: Controlled DeepSeek scale-up. Exact success (E) and core-order success (C) support different conclusions on the same outputs. Inv. is invalid source-rule transfer on contrastive examples.

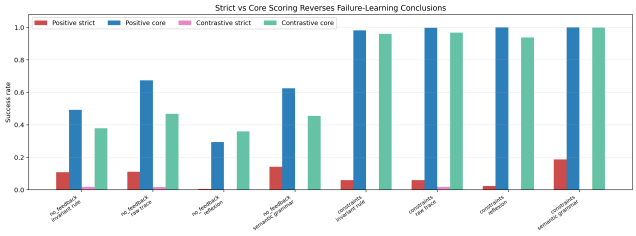

Figure 2: Exact success stays low even when core-order success is high.

calls, and 34.68M tokens. A subsequent balanced audit adds 11,520 rows: 2,880 for each of DeepSeek Chat, DeepSeek Reasoner, Qwen2.5-7B, and Mistral-7B-v0.3. Its grid crosses two feedback modes, five baselines, three splits, eight source tasks, and 12 seeds. We report task-cluster bootstrap intervals (5,000 resamples) so repeated rows do not masquerade as independent task breadth. The supplement gives the full tables, model settings, accounting, and an earlier token-cost stress slice.

Claim-Level Results

The same outputs support opposite abstracts. Table 2 shows the central pathology. Under constraint feedback, positive core transfer reaches 0.982–1.000 across baselines, but exact positive transfer remains 0.023–0.187. The contrastive split is sharper: core target-solving reaches 0.939–0.999 while exact contrastive success remains 0.000–0.018. An exact-only paper would report that the agents mostly fail to learn from failure. A core-only paper would report that they almost always succeed. Both conclusions are incomplete.

Verifier calibration is an experimental variable. Constraint feedback changes core success much more than exact success. Relative to no feedback, it raises positive core transfer by 0.324–0.705 and contrastive core target-solving by 0.500–0.581. Exact positive transfer changes only by -0.052 to +0.045, and exact contrastive success remains near zero. Calibration therefore changes what the evaluation measures. It does not merely make the model better by a small margin.

Invariant prompting is a necessary baseline. Invariant prompting is not uniformly best, but it is too strong to omit. Under calibrated positive transfer, the invariant baseline reaches 0.982 core success; under calibrated contrastive target-solving, it reaches 0.960. These values are close to raw trace and semantic grammar and above Reflexion-style prompting on contrastive target-solving. A memory or reflection method that beats only a weak prompt baseline has not shown that episodic learning caused the gain.

Contrastive failure is not mainly overtransfer. The controlled suite and executable workflow adapter both weaken a tempting overgeneralization story. Invalid source-rule transfer is essentially zero in calibrated controlled runs and is exactly zero in the executable workflow adapter. Contrastive difficulty mostly reflects target-task solving and hidden auxiliary actions, not rampant copying of source repairs into incompatible targets. This negative result is important because it narrows the paper's claim. The supported claim is metric decomposition, not a universal statement that agents overtransfer failure rules.

The conclusion flips at the claim level. The disagreement is not merely numerical. We label each mode-baseline-split row as success, mixed, or failure under exact and core reporting. Across 16 controlled rows, all 16 labels change, and the 8 calibrated rows flip from exact failure to core success. This audit is intentionally local rather than a published-paper reproduction, but it shows the kind of claim instability that a published evaluation could hide: the same outputs can support mutually incompatible abstracts depending on the reported score.

Executable Evidence

The strongest objection to core-order scoring is that it may be too relaxed. We address this by pairing core scoring with operation-vocabulary checks and executable validation.

Local executable checks. A dependency checker recomputes strict and dependency-valid core success from the downloaded JSONL outputs. It validates 41,178 rows and finds 21 disagreements, all caused by operations outside the declared task vocabulary in one no-feedback raw-trace contrastive group. This supports core-order scoring as a structural metric while also showing why it must be paired with an allowed operation or executable guard. A small unit-test repair adapter demonstrates both directions: core-minimal repairs can pass tests while failing exact matching, but outputs with unknown extra operations can satisfy bare core order while failing execution.

Real-LLM tool-use workflows. We then run a larger executable adapter in which DeepSeek generates tool-call plans for workflows such as reserving inventory before charging, snapshotting state before deletion, authorizing before rewriting a payload, and checking service health before announcing readiness. The simulator validates the final state and records exact, core, executable, hidden-action, vocabulary, and invalid-transfer metrics.

In this 92,160-row run, exact, core, and executable metrics again disagree substantially. The maximum core-minus-exact gap is 0.628, and 1,417 rows are core-valid but executable-invalid, mostly because an auxiliary operation appears before

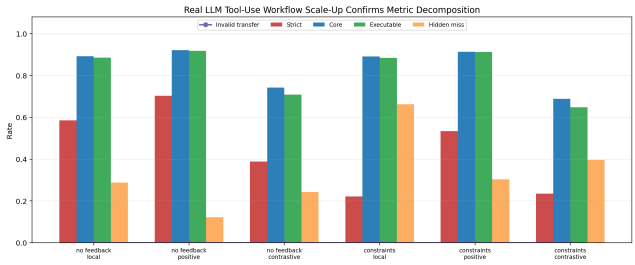

Figure 3: In the real-LLM workflow run, exact, core, and executable metrics disagree, while invalid source-rule transfer is absent.

Mode	Split	Rows	Exact	Core	Exec	Inv.
None	local	15360	.718	.918	.904	.000
None	positive	15360	.290	.752	.717	.000
None	contrastive	15360	.389	.742	.708	.000
Constraints	local	15360	.729	.935	.918	.000
Constraints	positive	15360	.293	.751	.704	.000
Constraints	contrastive	15360	.235	.689	.648	.000

Table 3: Executable workflow scale-up. Rows are DeepSeek-generated tool-call plans scored by exact, core, executable, and invalid-transfer metrics.

its own precondition. Exact matching is therefore too narrow, but core-order success alone is too permissive. The right object is the stack.

The strict–core gap replicates across model families. Table 4 reports a balanced audit over the same task and scoring interface. Every configuration completes 2,880 rows. Pooled strict–core gaps range from 0.393 to 0.591, while core–execution gaps range from 0.000 to 0.031. Thus exact scoring is consistently the larger source of claim reversal, whereas the added value of execution is real but model-dependent. The contrastive raw-trace/no-feedback cluster illustrates the uncertainty scale: DeepSeek Chat reaches .604 exact $[.510, .698]$, .781 core $[.729, .833]$, and .729 executable $[.667, .781]$; DeepSeek Reasoner under constraint feedback reaches .458 $[.344, .562]$, .833 $[.771, .896]$, and .771 $[.667, .875]$, respectively. Intervals resample source tasks.

The audit also falsifies the stronger overtransfer hypothesis. Invalid source-rule transfer occurs once in 11,520 rows, in a DeepSeek Reasoner output, and never in the other three configurations. The robust finding is therefore the need to separate strict, semantic, and executable claims—not a universal tendency to copy a failed source rule.

Protocol Audits and Claim Boundaries

The metric stack is intended to constrain scientific interpretation, so we audit both the scorer and the evidence pipeline. These checks address three failure modes that a single aggregate score cannot expose: an over-specified gold plan, a relaxed scorer that credits unsafe extra actions, and a run ledger that reports intended jobs rather than completed model outputs.

Model	Rows	Strict	Core	Exec	C–S	C–E
DS-Chat	2880	.441	.834	.818	.393	.016
DS-Reasoner	2880	.376	.845	.822	.469	.023
Qwen2.5-7B	2880	.217	.808	.808	.591	.000
Mistral-7B	2880	.202	.749	.719	.548	.031

Table 4: Balanced cross-model audit. C–S and C–E are the core-minus-strict and core-minus-executable gaps. Pooled rates are descriptive; inference uses source-task clusters.

Candidate repair	Exact	Core	Executable
Core minimal	.000	1.000	1.000
Failed original	.000	.000	.000
Reversed core	.000	.000	.000
Strict gold	1.000	1.000	1.000
Unknown extra operation	.000	1.000	.000

Table 5: API-free unit-test adapter. Core-minimal repairs can be valid without exact matching, while unknown extra operations require an executable guard.

Operational definitions. Local repair is evaluated on the source instance before any transfer claim is made. Positive transfer then evaluates a lexically changed target that preserves the source invariant. Contrastive target-solving evaluates a nearby target with a different invariant against that target's own gold plan. Invalid transfer is counted only when the source repair is copied or prefixed where it is wrong for the contrastive target. Hidden-action miss is diagnostic rather than a failure label: it records omission of actions declared optional by the task. This separation prevents a correct contrastive solution from being misreported as non-transfer and prevents a target-solving failure from being misreported as source-rule overgeneralization.

The distinction can be seen in a concrete workflow. Suppose the gold repair contains `normalize_schema`, `rollback_state`, and `insert_guard`, but the task text requires only normalization before the guard. The two-action prediction is exact-false but core-valid. If a plan also calls an undeclared `email_admin` operation, bare core order can still be true although vocabulary and execution checks reject the plan. On a contrastive target whose correct dependency is snapshot before deletion, a snapshot-delete prediction is target-solving; reserve-before-charge is invalid source transfer. The four outcomes imply different mechanisms and therefore must not share one label.

Scorer audit. We independently recompute exact and dependency-valid core scores from the downloaded controlled-run JSONL files. Across 41,178 rows, the recomputation finds 21 stored-score disagreements. All occur in one no-feedback, raw-trace, contrastive group and all contain an operation outside the task's declared vocabulary. Treating these rows conservatively as executable failures produces a 0.673 rate of strict-false but dependency-valid outputs over the controlled suite. Thus the large exact/core gap is not explained by a broken subsequence implementation, but the 21 cases demonstrate why core order needs the vocabulary

Evidence quantity	Verified count
Controlled valid rows	41,178
Controlled API calls	82,371
Controlled tokens	55.66M
Workflow valid rows	92,160
Workflow API calls	92,161
Workflow tokens	34.68M
Balanced cross-model rows	11,520
Chat stress rows / tokens	11,520 / 4.33M
Reasoner stress rows / tokens	4,190 / 8.33M

Table 6: Evidence ledger counted from valid row-level outputs. Marker files and intended jobs are non-authoritative.

check included in the proposed reporting standard.

Run accounting as a validity control. The intended controlled scale-up comprised 168 chunks, but the account reached an HTTP 402 insufficient-balance response after roughly 28–29 effective chunks. The original asynchronous launcher could write later completion markers after background workers had already failed. We therefore discard marker-derived completion claims and derive every count from parseable JSONL rows. Table 6 reports the resulting ledger. The earlier reasoner stress lane is explicitly partial, whereas the cross-model audit is balanced. This is not merely an engineering note: using expected chunk counts would change the evidentiary denominator and could make an incomplete experiment appear balanced.

Task-family and calibration controls. The controlled suite spans code-debug, proof-repair, and workflow-ordering families. Each family supplies source, positive, and contrastive variants, and the positive targets change surface names while preserving the violated dependency. The non-leaky constraint condition states only a boundary such as operation A preceding B; it does not reveal the complete target plan. Accordingly, its large effect on core success is evidence about calibration to the dependency being measured, not evidence that the evaluator supplied the gold answer. Reporting no-feedback and constraint conditions side by side makes this sensitivity visible.

The invariant prompt is equally important as a causal control. Reflection and semantic-memory prompts contain more episodic structure, but a direct invariant prompt can already reach 0.982 positive and 0.960 contrastive core success under constraints. Gains over raw prompting therefore do not by themselves identify memory as the mechanism. A failure-learning study should compare against the simplest prompt that states the transferable abstraction, then measure whether episodic machinery adds repair, transfer, or executable value beyond that baseline.

Artifact-level reproducibility. The anonymous code-and-data package includes the controlled, workflow, and balanced cross-model row-level outputs; evaluation inputs; scorer implementations; API-free unit and workflow adapters; and scripts that regenerate the summary and task-cluster interval tables. Local adapters verify the exact/core/executable separation without paid API calls; the included row-level responses allow the principal scoring audits to be rerun without incurring new model costs. The technical supplement contains the complete condition tables, prompt-family descriptions, stress-slice breakdown, and representative scorer examples. No result in the main paper is inferred from completion markers alone.

Recommended reporting protocol. A minimal claim-control evaluation should state the scientific claim before choosing its primary score. For a local-repair claim, report source repair and an executable check where one exists. For positive transfer, report exact and core success on a target that changes surface form while preserving the scoped invariant. For controlled non-transfer, report target-solving and invalid source-rule transfer separately; low target accuracy is not evidence that the agent correctly withheld a rule. When optional actions exist, report their miss rate. Finally, include a direct invariant baseline and account for valid rows, retries, failures, and token cost. These fields form a profile rather than a weighted composite because no universal weight can decide whether exact reproduction, semantic validity, or safe execution is the intended behavior.

What would change the conclusion. The methodology claim would weaken if exact/core disagreements disappeared on less templated tasks, if executable checks agreed with exact matching whenever task intent was explicit, or if a published failure-learning conclusion proved stable under the complete control stack. Conversely, the current evidence does not justify claiming that agents generally overtransfer: calibrated invalid-transfer rates are near zero and the workflow adapter records none. Nor does it show that reflection is ineffective. Reflection changes several metric components, but the direct invariant baseline prevents those changes from being uniquely attributed to episodic learning. These are deliberate non-claims, not omissions.

Dependence and uncertainty. Rows produced from the same source task are not independent evidence about conceptual breadth. The balanced audit adds genuine cross-family replication, but only across eight source tasks and four invariants. We therefore bootstrap source tasks, not rows, and present pooled model rates as descriptive summaries. The earlier unbalanced chat/reasoner slice remains a token-cost stress test, not evidence that one decoding mode is intrinsically superior.

Safety interpretation. Executable validity is a task-specific oracle, not a proof of real-world safety. The workflow simulator checks declared preconditions, operation vocabulary, and final state for the benchmarked plans. It cannot detect an undeclared side effect or an incorrect world model. Its purpose is narrower: to demonstrate that a semantically relaxed score can accept outputs that a behavioral checker rejects. In domains with tests, simulators, proof checkers, or transaction invariants, those executable checks should replace or augment the adapter used here.

Practical auditability. The row-level release makes every aggregate in the main tables traceable to a model response,

Scientific claim	Minimum supporting evidence	Disqualifying ambiguity
Local repair	Source exact/core plus execution	Transfer is not tested.
Positive transfer	Changed target with same invariant	Surface replay remains possible.
Exact reproduction	Ordered equality with gold	Gold may be overspecified.
Semantic validity	Core dependency plus vocabulary	Extra operations may be unsafe.
Controlled non-transfer	Target solved and source rule not copied	Target failure mimics restraint.
Executable repair	Behavioral checker passes	Mechanism remains unidentified.
Learning from failure	Above controls plus non-episodic baseline	Prompt calibration explains gain.

Table 7: Claim-to-evidence matrix for interpreting post-failure improvement.

condition, and scoring record. Re-analysis requires no new API calls. Re-running generation does require model access and can be costly, so the artifact separates deterministic scoring from optional regeneration. This division also makes future scorer revisions auditable: researchers can apply a new exact, semantic, or executable criterion to the same frozen outputs and observe whether the paper-level claim changes.

Claim-to-evidence matrix. Table 7 states the minimum evidence needed for each interpretation. This matrix is deliberately asymmetric. For example, high contrastive target accuracy supports target-solving but does not establish that the source rule was considered and rejected. Conversely, a low invalid-transfer rate is meaningful only when the model also had enough target-solving ability for the contrastive test to be nontrivial. Executable success can validate a plan under the benchmark simulator, but it does not identify whether success came from reflection, a direct invariant, or another prompt feature.

Potential confounds. Prompt length, response verbosity, and access to the failure trace differ across baselines and can affect exact matching independently of learning. The paper therefore uses the baselines as diagnostic probes, not as a ranking of prompting algorithms. Constraint feedback also changes the information given to the model; it is a calibration condition, not a fair head-to-head claim that one model learned more. Model outputs can be correlated within a task template or decoding configuration, so the row count should be read as precision within the designed suite rather than thousands of independent conceptual tasks.

Error decomposition. Observed failures fall into four operational categories. A specification error occurs when the gold plan contains an auxiliary action not required by task intent. A semantic error violates the required dependency even after optional actions are removed. A vocabulary error introduces an operation outside the declared task interface. An execution error satisfies the named core order but violates another precondition or final-state constraint. Exact, core, vocabulary, and executable metrics isolate these cases in that order. Collapsing them into success/failure prevents a reviewer from determining whether a method improved understanding, formatting, or behavioral safety.

Use beyond plan ordering. The reporting logic is not tied to the specific operation vocabulary used in the experiments. In code repair, exact match can be replaced by patch identity, core validity by semantic or invariant checks, and execution by tests. In proof repair, the corresponding layers may be exact derivation, required lemma dependencies, and proof-checker acceptance. In tool agents, transaction or safety invariants provide the executable layer. What transfers is the rule that a metric must be named together with the scientific claim it licenses and the nearest alternative explanation it fails to exclude.

Decision rule for reviewers. When a paper claims learning from failure, a reviewer can apply three questions. Did the model repair the source before transfer was evaluated? Did it succeed on a genuinely changed target while a direct invariant baseline was available? Did the reported score distinguish semantic validity from executable behavior and invalid source-rule copying? A negative answer does not prove that the agent failed to learn, but it does mean that the experiment has not isolated that conclusion. This conservative logic is the central product of the claim-control framework.

Related Work

Reflection and experiential-learning methods show that agents can benefit from feedback, memory, and self-generated critique. Reflexion stores verbal feedback to guide later decisions (Shinn et al. 2023); ExpeL extracts natural-language lessons from successes and failures (Zhao et al. 2024); Self-Refine studies iterative self-feedback within a generation (Madaan et al. 2023); and Voyager uses executable skill memory in an open-ended environment (Wang et al. 2023). These works establish that feedback can improve later behavior. Our question is orthogonal: what exactly has an evaluation measured when it reports improvement after failure?

The claim-control view is also related to counterexample-guided synthesis and debugging, where failures are used to refine hypotheses or programs (Solar-Lezama 2008). The difference is that language-agent evaluations often use natural-language traces and flexible plans, so exact matching, semantic dependency validity, executable validity, and invalid transfer can diverge on the same output. This makes reporting discipline part of the scientific contribution.

Tool-interactive correction strengthens the case for separating feedback from validation. CRITIC uses external tools to verify and revise model outputs (Gou et al. 2024), while controlled studies show that intrinsic self-correction can fail or even degrade reasoning without reliable external feedback (Huang et al. 2024). These findings are not contradictory: they concern different feedback channels and evaluators. The claim-control surface makes that distinction explicit by ask-

ing whether a gain comes from a repair trace, a scoped constraint, an executable checker, or an invariant already stated in the prompt.

Agent benchmarks broaden evaluation from isolated generations to interactive trajectories. AgentBench covers diverse interactive environments (Liu et al. 2024), ToolLLM studies large-scale API use (Qin et al. 2024), and SWE-bench evaluates repository-level patches using real software issues and executable tests (Jimenez et al. 2024). Our workflow suite is smaller in environmental breadth and should not be read as a replacement for these benchmarks. Its purpose is controlled decomposition: the same stored response is scored under exact, core, vocabulary, and execution criteria so that a paper-level conclusion can be traced to the evaluator that licenses it.

Recent continual-learning evaluation for language agents similarly emphasizes controlled task streams and transfer-aware measurement (Shu et al. 2026). Our scope is complementary: rather than proposing a single transfer-gain score, we ask which repair, transfer, target-solving, and execution claims each metric can license.

This perspective is also related to benchmark validity. A benchmark can be reproducible at the level of files and scripts while still under-identifying the scientific construct named in the abstract. Exact plan matching is reliable but may measure conformity to one serialization; semantic scoring is flexible but can over-credit hallucinated actions; execution is behaviorally grounded but depends on the completeness of the simulator. Claim-control evaluation does not select one universally correct metric. It requires the paper to report the nearest competing interpretations and enough controls to eliminate those that are inconsistent with the observed outputs.

Metric implementations are themselves part of the experimental object. A change from equality to subsequence validity, a revision to optional actions, or a new executable precondition can alter the paper-level conclusion without changing one model response. Our artifact therefore preserves predictions with each scorer field and regenerates both pooled summaries and task-cluster intervals. Future scorer revisions can be applied to the same frozen outputs and reported alongside, rather than silently replacing, the original decision. This separation is especially useful for paid agent evaluations whose exact responses may be impractical to regenerate.

Discussion

The results should not be read as a leaderboard for prompting methods. Raw trace, Reflexion-style prompting, semantic grammar, and invariant prompting are probes that expose how much the conclusion depends on scoring and calibration. The important result is the profile, not the winner. A method that improves exact success but leaves hidden-action miss high has a different failure mode from a method that improves core success but produces executable-invalid extra operations.

This is why the title question matters. "The agent learned from failure" is not an atomic conclusion. It may mean local repair, invariant transfer, target-solving, calibration sensitivity, or safe executable behavior. The paper's role is to make those claims auditable before a method is credited with learning from its failures.

The most portable artifact is therefore not a new prompt. It is the reporting discipline in Figure 1 and Table 1. A paper may still choose exact success as its primary metric when exact reproduction is the intended behavior, or executable validity when tests are available. The problem is claiming failure learning from a single aggregate score whose implied scientific claim is unstated. Our recommendation is to name the claim first and then report the controls needed to support that claim.

The run-accounting failure is part of the same story. Long-running agent evaluations often use asynchronous workers, paid APIs, retries, and partial resume logic. A completion marker can be less trustworthy than the row-level outputs it is meant to summarize. In this project, valid row accounting changed the evidentiary base after a 402 balance failure. Agent evaluations need the same operational discipline they demand from agents.

Limitations

The evidence is strong enough for the metric-decomposition claim, but it has boundaries. The balanced audit spans DeepSeek, Qwen, and Mistral families, yet it contains only eight source tasks and four invariants; task-cluster intervals make this limitation visible rather than converting 11,520 rows into fictitious conceptual breadth. The 92,160-row executable adapter is purpose-built rather than a public benchmark such as SWE-style repair. The local claim-flip audit also does not reproduce a named published result. These are generality limits, not contradictions of the current claim.

The balanced audit uses deterministic decoding; seed identifiers vary prompts and targets. Its intervals therefore quantify source-task rather than decoding variation.

The paper also deliberately weakens an earlier hypothesis. The data do not show that agents generally fail to learn from failure, nor that they generally copy failed source rules into incompatible targets. In the executable workflow adapter, invalid source-rule transfer is absent. This makes the contribution narrower but more reliable: the evidence supports a reporting standard for failure-learning claims.

Conclusion

Failure-learning claims are unstable unless evaluations distinguish local repair, semantic transfer, exact matching, core-order validity, hidden-action miss, contrastive target-solving, invalid source-rule transfer, executable validity, invariant baselines, and reliable run accounting. The same outputs can make an agent look as if it failed or succeeded depending on which claim is scored. A claim-control evaluation does not replace existing agent-learning benchmarks; it makes their conclusions interpretable.

Generative-AI use. Generative AI systems assisted with research ideation, manuscript drafting and editing, code generation, experiment orchestration, and result organization. Human authors remain responsible for verifying all submitted content.

References

Gou, Z.; Shao, Z.; Gong, Y.; Shen, Y.; Yang, Y.; Duan, N.; and Chen, W. 2024. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. In *The Twelfth International Conference on Learning Representations*.

Huang, J.; Chen, X.; Mishra, S.; Zheng, H. S.; Yu, A.; Song, X.; and Zhou, D. 2024. Large Language Models Cannot Self-Correct Reasoning Yet. In *The Twelfth International Conference on Learning Representations*.

Jimenez, C. E.; Yang, J.; Wettig, A.; Yao, S.; Pei, K.; Press, O.; and Narasimhan, K. R. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *The Twelfth International Conference on Learning Representations*.

Liu, X.; Yu, H.; Zhang, H.; Xu, Y.; Lei, X.; Lai, H.; Gu, Y.; Ding, H.; Men, K.; Yang, K.; et al. 2024. AgentBench: Evaluating LLMs as Agents. In *The Twelfth International Conference on Learning Representations*.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; Welleck, S.; Majumder, B. P.; Gupta, S.; Yazdanbakhsh, A.; and Clark, P. 2023. Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; et al. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs. In *The Twelfth International Conference on Learning Representations*.

Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366.

Shu, Y.; Jiménez Gutiérrez, B.; Jonnalagedda, S. P.; Yao, Y.; Sun, H.; and Su, Y. 2026. AGENTCL: Toward Rigorous Evaluation of Continual Learning in Language Agents. arXiv:2606.02461.

Solar-Lezama, A. 2008. *Combinatorial Sketching for Finite Programs*. Ph.D. thesis, University of California, Berkeley.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv:2305.16291.

Zhao, A.; Huang, D.; Xu, Q.; Lin, M.; Liu, Y.-J.; and Huang, G. 2024. ExpeL: LLM Agents Are Experiential Learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Supplementary Material for Failure-Learning Claims Need Controls: A Claim-Control Evaluation of Language Agents

Anonymous submission

Appendix Overview

This supplement provides the evidence and protocol details behind the main paper's claim-control evaluation. The main paper keeps the submission argument focused: a single failure-learning score is not enough to determine what an agent learned from a failed trace. The supplement records the metric definitions, task construction, run accounting, full result tables, executable validation, and claim-boundary checks needed to audit that argument.

Metric Definitions

For each source-target pair, the harness records local and transfer outputs. Let p_l and p_t be the local and transfer predicted plans, g_l and g_t the corresponding gold plans, and O_l and O_t the optional action sets.

Local repair. Local exact success is $\mathbb{1}[p_l = g_l]$. Local core success is $\mathbb{1}[c(g_l, O_l) \sqsubseteq p_l]$. In all reported LLM rows, local core repair is high enough that transfer and contrastive interpretation are meaningful.

Positive transfer. Positive exact transfer is $\mathbb{1}[p_t = g_t]$ on a target with the same invariant as the source. Positive core transfer is $\mathbb{1}[c(g_t, O_t) \sqsubseteq p_t]$.

Contrastive target-solving. Contrastive exact and core success use the target gold plan, not the source plan. A high contrastive score means the model solved the target. It does not mean invalid transfer.

Invalid source-rule transfer. Invalid transfer is counted when the transfer plan copies or prefixes the source repair in a target where that source repair is invalid. This is reported separately because many models fail contrastive tasks by not solving the target, not by copying the source rule.

Hidden-action miss. Hidden-action miss is computed only for declared optional actions. It is not a failure score by itself. It diagnoses how often core-valid responses omit auxiliary actions.

Task Families and Prompt Conditions

The controlled suite covers three families: code-debug ordering, proof-repair ordering, and workflow ordering. Each family defines source, positive, and contrastive target variants. The positive target preserves the violated invariant while changing names or surface context. The contrastive target is a near-neighbor with a different correct invariant.

The prompting baselines are deliberately simple probes rather than proposed new algorithms.

- **Raw trace** gives the agent the local failure and repair context.
- **Reflexion-style reflection** asks the agent to summarize the lesson from the failure before transfer.
- **Semantic grammar** asks for structured fields: precondition, violated invariant, forbidden transfer, repair ordering, witness, scope, and non-scope.
- **Invariant rule** asks directly for the scoped invariant. It is a missing-baseline condition: if it is competitive, memory or reflection gains cannot be attributed solely to episodic learning.

The two main feedback modes are no-feedback transfer and non-leaky constraint feedback. Constraint feedback states a dependency boundary, such as operation A needing to precede operation B, but does not reveal the full ordered target plan.

Controlled Scale-Up Results

The controlled scale-up contains 41,178 valid rows, 82,371 API calls, and 55.66M tokens. Table 1 is the main strict-versus-core result with row counts.

Table 1: Controlled DeepSeek scale-up with counts. Exact and core-order success support different conclusions on the same outputs.

Mode	Baseline	Rows	Pos-E	Pos-C	Ctr-E	Ctr-C	Inv.
None	Invariant	5148	.109	.492	.018	.379	.000
None	Raw trace	5148	.111	.674	.016	.468	.007
None	Reflexion	5148	.006	.295	.000	.360	.000
None	Semantic grammar	5148	.142	.626	.001	.455	.000
Constraints	Invariant	5148	.059	.982	.001	.960	.000
Constraints	Raw trace	5148	.060	.998	.018	.968	.000
Constraints	Reflexion	5148	.023	1.000	.000	.939	.000
Constraints	Semantic grammar	5142	.187	1.000	.001	.999	.000

Calibration gain. Relative to no feedback, constraints improve positive core transfer by 0.324–0.705 and contrastive core target-solving by 0.500–0.581. Exact positive transfer changes only by -0.052 to +0.045, and exact contrastive success remains near zero.

Executable Validation

The strict/core decomposition would be weak if core-order scoring merely credited incomplete or unsafe plans. We therefore add operation-vocabulary and executable checks.

Downloaded-row scorer check. A dependency checker recomputes exact and dependency-valid core success from the downloaded JSONL outputs. It validates 41,178 rows and finds 21 disagreements with the stored core score. All 21 are cases where the model used an operation outside the declared task vocabulary. This supports the structural core scorer while justifying the operation-vocabulary caveat.

Unit-test repair adapter. Table 2 summarizes a local adapter where candidate repairs are validated by tests rather than exact plan matching.

Candidate	Exact	Core	Unit test
Core minimal	0.000	1.000	1.000
Failed original	0.000	0.000	0.000
Reversed core	0.000	0.000	0.000
Strict gold	1.000	1.000	1.000
Unknown extra	0.000	1.000	0.000

Table 2: Local executable repair adapter. Core-minimal repairs can be valid without exact matching, but hallucinated extra operations require an executable or vocabulary guard.

Real-LLM Tool-Use Workflow Scale-Up

The workflow adapter models API/tool plans rather than symbolic plan strings. Each task involves dependencies such as reserving inventory before charging, snapshotting state before deletion, authorizing before rewriting a payload, or checking service health before announcing readiness. The simulator validates whether the final state is safe and records exact, core, executable, hidden-action, vocabulary, and invalid-transfer metrics.

Table 3: Real-LLM workflow scale-up by mode and split. Strict, core, and executable metrics support different interpretations, while invalid source-rule transfer remains zero.

Mode	Split	Rows	Exact	Core	Exec	Inv.
None	local	15360	.718	.918	.904	.000
None	positive	15360	.290	.752	.717	.000
None	contrastive	15360	.389	.742	.708	.000
Constraints	local	15360	.729	.935	.918	.000
Constraints	positive	15360	.293	.751	.704	.000
Constraints	contrastive	15360	.235	.689	.648	.000

The maximum core-minus-exact gap is 0.628. The run contains 1,417 core-valid but executable-invalid rows, mostly because an auxiliary operation appears before its own precondition. Thus exact matching is too narrow, but bare core order is also too permissive.

DeepSeek-Chat and DeepSeek-Reasoner Stress Slice

This earlier stress slice uses the same workflow setup for DeepSeek-chat and DeepSeek-reasoner, but it should be interpreted as a cost-controlled pressure test rather than a fully symmetric benchmark. Chat completes 11,520 valid rows using 4,332,457 tokens, or 376 tokens per valid row. Reasoner returns 4,190 valid rows before the 13-hour wall-clock limit, using 8,328,767 tokens, or 1,988 tokens per valid row. Thus the reasoner spends about 5.3x as many tokens per valid row. Invalid source-rule transfer remains essentially absent (0 for chat and 0.0005 for reasoner), but the additional reasoning budget does not remove the contrastive transfer failures.

Table 4: High-signal contrastive rows from the chat/reasoner stress slice. The n column differs because the reasoner lane timed out before completing the full grid. Invalid source-rule transfer is zero for all shown conditions.

Model	Condition	n	Exec	Core	Hidden
Chat	Raw trace, no feedback	384	.763	.807	.099
Reasoner	Raw trace, no feedback	139	.647	.770	.165
Chat	Raw trace, constraints	384	.690	.755	.268
Reasoner	Raw trace, constraints	139	.640	.705	.381
Chat	Sem. grammar, constraints	384	.661	.685	.422
Reasoner	Sem. grammar, constraints	139	.540	.561	.460

Balanced Cross-Model Audit

We subsequently run a balanced grid over DeepSeek Chat, DeepSeek Reasoner, Qwen2.5-7B, and Mistral-7B-v0.3. Each configuration contributes 2,880 valid rows, for 11,520 total:

$$4 \text{ configurations} \times 2 \text{ feedback modes} \times 5 \text{ baselines} \times 3 \text{ splits} \times 8 \text{ source tasks} \times 12 \text{ seeds}.$$

The eight source tasks instantiate four invariants. Because the 12 seeded responses for a source task do not create 12 independent concepts, uncertainty is estimated with a nonparametric source-task bootstrap using 5,000 resamples. The random seed for the bootstrap and all row-level inputs are included in the artifact.

Models, decoding, and infrastructure. The API configurations are `deepseek-chat` and `deepseek-reasoner`; the open-weight checkpoints are `Qwen/Qwen2.5-7B-Instruct` and `mistralai/Mistral-7B-Instruct-v0.3`. API decoding uses temperature zero, and local decoding is greedy with at most 192 new tokens. Thus the 12 seeds vary prompt wording and target pairing; they are repetitions within a source-task cluster, not independent sampling draws. Local models run in bfloat16 with PyTorch 2.5.1 (CUDA 12.1) and Transformers 4.46.3 on NVIDIA RTX 4090 GPUs. The host uses Ubuntu 22.04, 64 logical AMD EPYC 7282 CPUs, and 251 GiB RAM. The artifact records the exact commands and model identifiers.

Model	Rows	Strict	Core	Exec	C–S	C–E
DeepSeek Chat	2880	.441	.834	.818	.393	.016
DeepSeek Reasoner	2880	.376	.845	.822	.469	.023
Qwen2.5-7B	2880	.217	.808	.808	.591	.000
Mistral-7B-v0.3	2880	.202	.749	.719	.548	.031

Table 5: Pooled balanced-audit rates. C–S and C–E denote core-minus-strict and core-minus-executable gaps. These rates summarize the designed grid; clustered intervals below quantify task-level uncertainty.

Table 6: Representative source-task-clustered 95% bootstrap intervals. Each row contains 96 responses over eight source tasks.

Model	Split	Feedback	Baseline	Strict	Core	Exec
DS Chat	Ctr.	None	Raw	.604 [.510,.698]	.781 [.729,.833]	.729 [.667,.781]
DS Reasoner	Ctr.	Constraints	Raw	.458 [.344,.562]	.833 [.771,.896]	.771 [.667,.875]
Mistral-7B	Pos.	None	Sem. grammar	.177 [.042,.344]	.823 [.667,.958]	.823 [.656,.958]
Qwen2.5-7B	Pos.	None	Raw	.646 [.375,.896]	1.000 [1.000,1.000]	1.000 [1.000,1.000]

The exact/core disagreement replicates across all four configurations: core-minus-strict ranges from .393 to .591. By contrast, core-minus-executable ranges from .000 to .031, so executable checking adds a smaller and model-dependent correction in this audit. The inference is not that execution can be omitted—the 92,160-row workflow run contains 1,417 core-valid but executable-invalid plans—but that the magnitude of this second boundary depends on the output distribution.

Only one of 11,520 balanced-audit rows is classified as invalid source-rule transfer. It comes from DeepSeek Reasoner; the other configurations have zero. This result rejects the paper's initial broad overtransfer hypothesis while strengthening the narrower claim-control result. Exact, core, and executable metrics continue to license different conclusions across model families, even when the invalid-transfer mechanism is nearly absent.

Run Accounting and API Failure

The controlled run exposed a reproducibility issue common to long-running API evaluations. The intended scale-up had 168 chunks, but the DeepSeek account hit a 402 insufficient-balance error after about 28–29 effective chunks. The original launcher could write later marker files even after background workers failed. All evidence in the paper is therefore counted from valid JSONL rows, not from intended jobs or marker files.

Quantity	Count
Controlled valid rows	41,178
Controlled API calls	82,371
Controlled tokens	55.66M
Workflow valid rows	92,160
Workflow API calls	92,161
Workflow tokens	34.68M
Balanced cross-model valid rows	11,520
Rows per balanced configuration	2,880
Source tasks / invariants	8 / 4
Cluster bootstrap resamples	5,000
Chat stress valid rows	11,520
Reasoner stress valid rows	4,190
Chat stress tokens	4,332,457
Reasoner stress tokens	8,328,767
Chat tokens per valid row	376
Reasoner tokens per valid row	1,988

Table 7: Run accounting used by the paper. Valid rows, not marker files, define the evidence base.

Threat Model and Scorer Examples

Local repair illusion. A system may repair the source example without learning a transferable rule. Local repair should therefore be a prerequisite for transfer analysis, not the transfer metric itself.

Exact gold over-specification. A gold plan may contain auxiliary actions that are useful but not necessary, or actions that are only implicit in the task text. Exact scoring can then underestimate meaningful repair.

Relaxed over-credit. A core-order metric can over-credit outputs that contain the right dependency plus hallucinated or invalid operations. This is why core scoring must be paired with vocabulary or executable validation.

Target-solving versus transfer. A model can solve a contrastive target without transferring the source rule. This is evidence that the model understands the target, not invalid transfer.

Representative examples. If the gold repair contains `normalize_schema`, `rollback_state`, and `insert_guard`, but the task text only requires normalization before the guard, then a prediction with `normalize_schema` followed by `insert_guard` is exact-false but core-valid. If the required dependency is snapshot before deletion, then a plan that snapshots and deletes but also calls an undeclared `email_admin` operation may be core-valid but executable-invalid. If the source rule is reserve before charge and the contrastive target is snapshot before delete, then a snapshot-delete prediction is contrastive target-solving, while a reserve-charge prediction is invalid source transfer.

Artifact and Reproducibility Controls

Table 8: Current reproducibility controls.

Control	Current status
Row-level accounting	Valid JSONL rows counted directly; markers are non-authoritative.
API failure handling	Failure observed and documented; patched launchers should wait on worker exits before marker creation.
Rule diversity	Canary and scale-up reports include distinct rule ids and rule-pair ids.
Exact and core metrics	Both reported in main tables.
Hidden-action diagnostics	Reported separately, not folded into success.

Control	Current status
Contrastive target-solving	Reported as target success, not invalid transfer.
Invalid transfer	Measured as source-rule copying into a wrong target.
Operation vocabulary	Locally validated; unknown-operation caveat retained.
Executable validation	Demonstrated on unit-test and 92,160-row workflow adapters.
Model-family replication	Balanced 11,520-row audit across DeepSeek, Qwen, and Mistral families.
Dependence control	Confidence intervals resample eight source tasks rather than 11,520 rows.

Claim Boundary

The paper should not claim that language agents generally fail to learn from failure. It should not claim that Reflexion-style memory is useless. It should not claim that invariant prompting always wins. It should not claim that exact scoring is wrong in all cases or that core-order scoring is sufficient by itself. The supported claim is narrower and stronger: failure-learning conclusions are unstable unless evaluations separately report local repair, exact transfer, core transfer, hidden-action miss, contrastive target-solving, invalid source-rule transfer, invariant baselines, executable validity, and reliable run accounting.

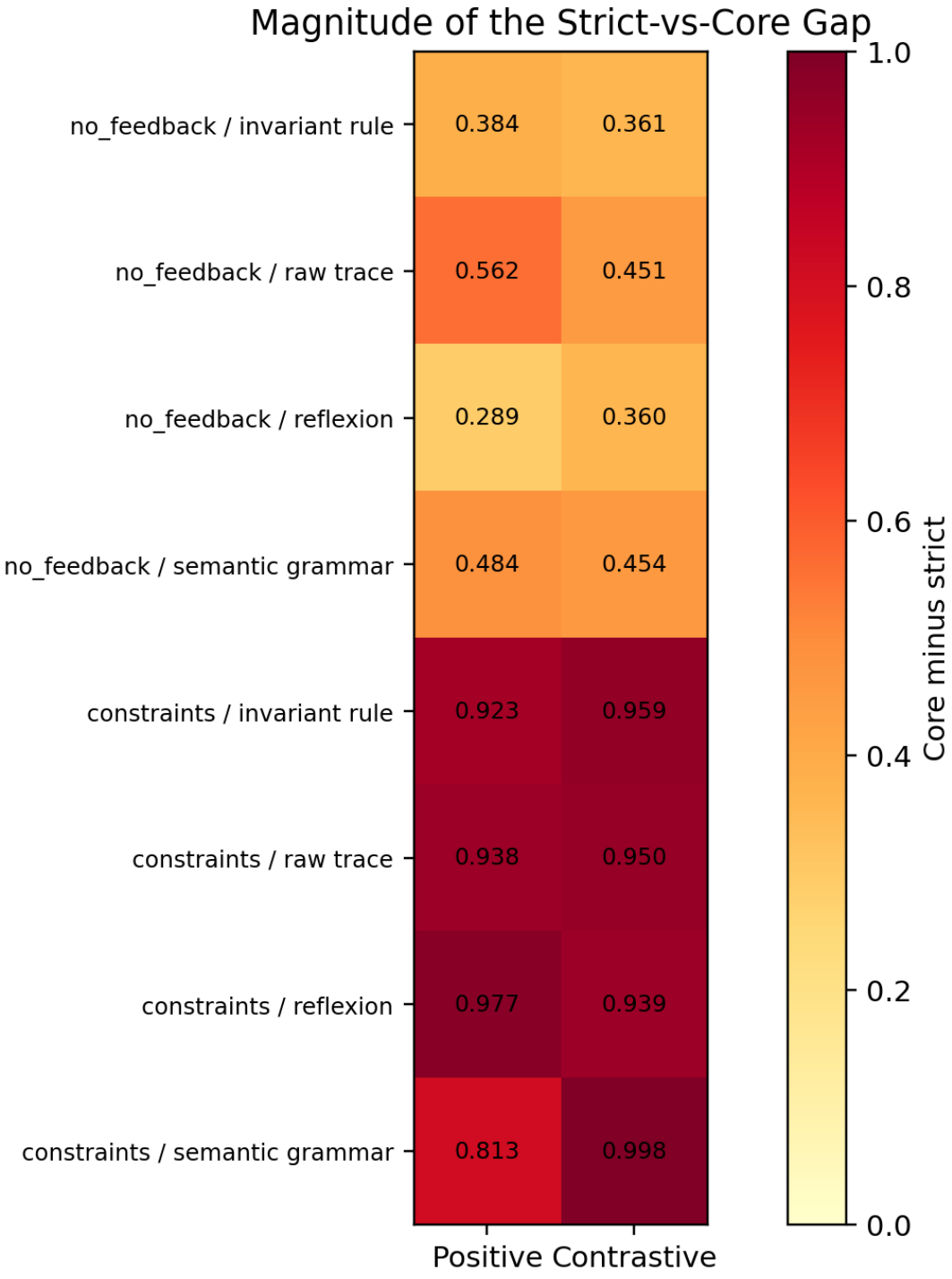

Figure 1: The exact-versus-core gap is largest under calibrated constraint feedback.

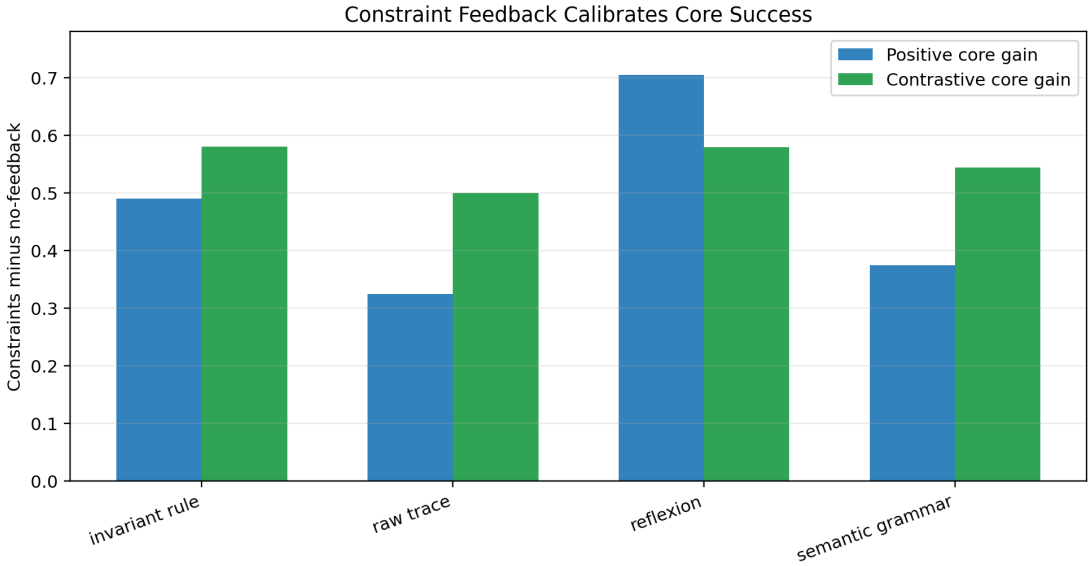

Figure 2: Constraint feedback substantially improves core success while exact success remains mostly low.

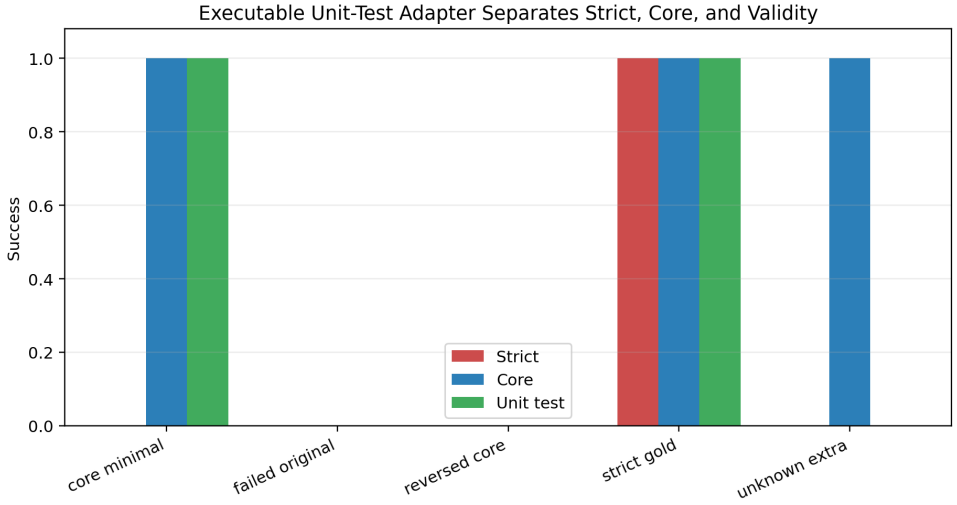

Figure 3: Executable validation separates exact-plan success, core-order success, and unit-test validity.

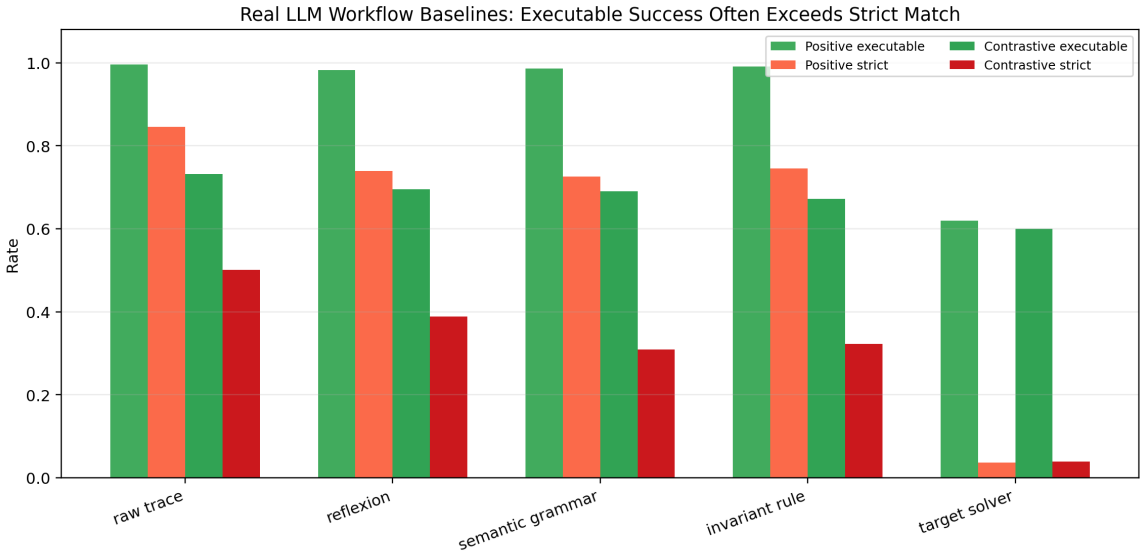

Figure 4: Executable success often exceeds exact-plan success, but executable validation remains necessary because core-valid plans can still be unsafe.

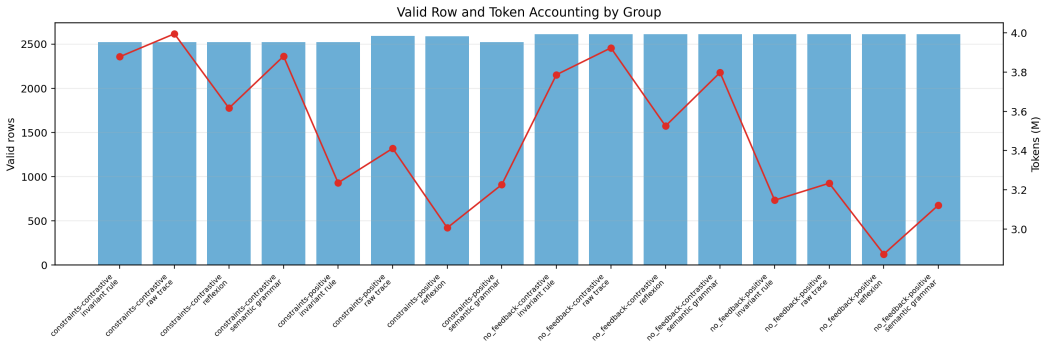

Figure 5: Progress accounting by valid rows and tokens. Expected completion markers overstated the controlled run after API balance failure.