ATTRIBUTING FAILURES IN EXECUTABLE LANGUAGE MODEL REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Quantitative reasoning with language models increasingly runs through an executable proxy: the model writes a program, symbolic expression, or tool call, and a deterministic system produces the answer. When such a pipeline underperforms unrestricted language-model reasoning, end-to-end accuracy does not reveal which stage is responsible. We study this attribution problem with controlled substitutions that isolate representation construction, deterministic execution, and answer evaluation while holding budget and scoring fixed. Across four established reasoning datasets and two model families, superficially similar gaps resolve into different causes: a TabMWP gap of 0.117 disappears at matched generation budget, and most of a TAT-QA gap traces to a scorer-side percent convention. A complete census of frozen baseline failures separates crash-type (execution-invalid) errors from executable-but-semantically-wrong ones. A pre-registered prospective study then shows that attribution predicts intervention response: repair gains are concentrated in the prospectively diagnosed execution-invalid class (interaction $\Gamma = 0.62$ $[0.40, 0.82]$; cross-model $\Gamma = 0.51$ $[0.39, 0.63]$; pooled $\Gamma = 0.50$ $[0.40, 0.61]$, cluster bootstrap, over 460 question-condition evaluations spanning 160 problems), with zero repair-induced errors. A matched self-debugging baseline reproduces the effect, locating the mechanism in the execution-feedback intervention class. Execution-valid semantic errors require different evidence and interventions.

1 INTRODUCTION

Executable reasoning systems translate language into programs, symbolic expressions, tool calls, or other executable representations, and a deterministic executor produces the final answer (Gao et al., 2023; Chen et al., 2023; Gou et al., 2024). The division of labor is attractive: language models handle interpretation, and the executor guarantees the computation.

An end-to-end performance difference in such systems is ambiguous, because several interfaces contribute to the final answer. The model may have chosen the wrong operands or operation. It may have produced malformed output under a tight generation budget. The executor may reject a valid construct. The scorer may misread a correct value. Accuracy alone cannot separate these, and method comparisons built on it can credit or blame the wrong component. Attribution matters operationally because a useful diagnosis should predict which intervention can change the outcome; the prospective experiment in this paper asks whether it does.

A concrete example: on TabMWP, our executable baseline scored 0.850 against a hand-verified oracle representation at 0.967 — an apparent representation-construction gap of 0.117. Matching the generation budget (800→1200 output tokens) closes the gap entirely: the parser reaches 0.967, equal to the oracle. The gap was truncation, not representation ability. A second example: on TAT-QA, the parser's apparent 0.15 gap shrinks to 0.016 when the scorer stops double-converting the model's own percent factors. End-to-end comparison alone would have misattributed both.

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.

Failure attribution in executable reasoning

Given a pipeline

$$x \xrightarrow{g_\theta} z \xrightarrow{E} y \xrightarrow{S} \hat{a},$$

where the model g_θ constructs an executable representation z, a deterministic executor E computes $y = E(z)$, and a scorer S produces the final answer, end-to-end accuracy alone cannot identify whether an error entered through representation construction g_θ, execution E, or evaluation S.
We test an attribution by intervening on the corresponding interface and checking whether the predicted response appears:

$$\text{diagnosed failure present} \implies \Delta_{\text{intervention}} > 0, \qquad \text{diagnosed failure absent} \implies \Delta_{\text{intervention}} \approx 0,$$

with the interaction $\Gamma = \Delta(E_1) - \Delta(E_0)$ measuring how selectively an intervention acts on its diagnosed failure class.

We use this strategy across four established reasoning datasets (TAT-QA, TabMWP, MATH, DROP) and two model families. The instruments are: a label-blind oracle substitution for representation construction, matched generation budgets, versioned scorers with both policies reported where the policy matters, a complete census of frozen baseline failures, and a pre-registered prospective study in which a failure mode is first diagnosed per question and then intervened on.

Contributions.

1. **Intervention-based attribution.** A controlled framework for localizing failure sources in executable reasoning: label-blind oracle substitution, matched budgets, versioned scorers, and a complete failure census, with every number traceable to frozen caches (Sec. 3, Table 1, Sec. 5).
2. **Prospective predictive validity.** A pre-registered study assigns each representation a failure diagnosis before any intervention; the diagnosis identifies where execution-feedback repair has recoverable opportunity (pooled interaction $\Gamma = 0.50$ $[0.40, 0.61]$, 460 evaluations over 160 problems, replicated across model families and budget conditions) (Sec. 7).
3. **Boundaries of execution feedback.** Matched controls and semantic-error experiments characterize what execution feedback does and does not expose: a matched self-debugging baseline reproduces the repair effect, and execution-valid semantic errors remain largely uninstrumented (Sec. 6, Sec. 8).

2 Related work

Program-aided and executable reasoning. PAL (Gao et al., 2023), Program-of-Thoughts (Chen et al., 2023), and ToRA (Gou et al., 2024) delegate computation to an interpreter; ReTool (Feng et al., 2025) trains tool use with reinforcement learning. These works ask whether executable reasoning raises end-task accuracy, evaluated against direct prompting on the same benchmarks; we ask where in the pipeline an observed gap enters.

Execution feedback and self-correction. Self-Debugging establishes execution feedback as an effective mechanism for repairing failed generations (Chen et al., 2024); Reflexion stores verbal feedback across attempts (Shinn et al., 2023); ReVeal generates tests as self-verification for code agents (Zhou et al., 2026); ReTool trains tool use through reinforcement learning (Feng et al., 2025). We use the same class of feedback as an intervention and evaluate whether an execution-validity diagnosis available beforehand identifies examples with recoverable error. The matched Self-Debug comparison in our study yields the same recovery pattern, tying the effect to the diagnosed failure class and to execution feedback rather than to a particular repair implementation. EVER (Zhang et al., 2026) uses LLM-mediated inspection of execution results to correct structural and semantic deviations in KBQA semantic parsing; Sec. 8 adapts its semantic correction to our frozen residual failures.

Evaluation of reasoning pipelines. Verifier-based selection (Lightman et al., 2024) and self-consistency (Wang et al., 2023) judge among generated outputs (learned or vote-based, but always at

the answer level); test-time-compute analyses compare sampling and search strategies by their end-task scores (Snell et al., 2025). Both evaluate at the pipeline's output. Our controls target the layer underneath: budget, executor, and scorer confounds that end-task comparisons inherit. A method that wins an end-task comparison may have won the harness; our artifact results give two concrete instances where the harness, not the model, produced the measured gap.

3 Problem formulation and attribution controls

Pipeline. Given question q and evidence e: $z = g_\theta(q, e)$ (representation: JSON arithmetic expression or sympy program), $y = E(z)$ (deterministic executor), $\hat{a} = S(y)$ (scorer/normalization). Arms on identical inputs: A_L (direct LM answer), A_{PX} (the executable pipeline), A_{OPX} (label-blind hand-verified representation through the same E and S). A no-evidence arm A_0 (the model answers without seeing the evidence) bounds what prior knowledge contributes: it is near zero on TAT-QA (0.008) and DROP (0.067), and 0.45–0.59 on MATH, where many answers are guessable from problem form. On the evidence-dependent tasks, then, every measured gap is an evidence-processing gap; on MATH, part of every score is prior knowledge, which is why the interaction test matters more than levels. $V_P = A_{\mathrm{OPX}} - A_{\mathrm{PX}}$ upper-bounds the loss attributable to representation construction, and $V_R = A_L - A_{\mathrm{PX}}$ measures whether executing at all helps over direct reasoning.

Controls. *Matched budget*: all arms share a 1200-token generation budget with one regeneration on empty output, unless a control varies it deliberately. *Matched scorer*: one versioned canonical scorer per dataset; where the scorer embodies a policy (percent-scale conversion, LaTeX normalization), both policies are reported. *Oracle discipline*: oracle representations are written label-blind and pass through the same executor and scorer; the two oracle construction errors are retained and counted against the oracle. *Frozen subsets*: 30–100 questions per split, fixed before measurement, with sample-id hashes (Appendix B). The representations themselves are minimal: for table tasks, a JSON object {`answer_type, expression, spans`} where the expression ranges over literals copied from the evidence (e.g., `(1609000-1164000)/1164000`); for MATH, a self-contained sympy program ending in `print(answer)`. Nothing in the representation format is tuned per dataset beyond the schema and the executor.

Freezing and reproducibility. Every model response is cached with its prompt hash, and every reported number regenerates from these caches without new model calls. Sample splits are drawn with fixed seeds before measurement, and the prospective gate sample is disjoint from the earlier evaluation subsets.

4 Main attribution results

Table 1 carries the paper's central finding: the same kind of observation (executable pipeline below a reference) resolves into different causes once the corresponding control is applied. Table 2 gives the canonical numbers.

Two readings of Table 1 matter. First, the naive gaps were all in the same range (0.12–0.25), and the causes are all different. Second, the controls are cheap: matching a budget, fixing a percent convention, or reading failures individually changes the conclusion before any model improvement.

TabMWP: budget creates apparent gaps. At the an 800-token budget the parser scored 0.850 against the oracle's 0.967. At the matched 1200-token budget the parser reaches 0.967, exactly the oracle: the entire 0.117 gap was silent truncation (14/60 responses cut off), not representation difficulty. Representation-generation budget can create an apparent pipeline gap.

TAT-QA: scoring policy moves the gap. Under the canonical percent convention (executor multiplies percent answers by 100), the parser scores 0.700/0.667 against the oracle's 0.900/0.767. Under an idempotent policy that does not double-convert the model's own `*100` factors, the parser scores 0.800/0.833 while the oracle is unchanged: the gap shrinks from 0.150 to 0.016 on the pooled calculable set. On the confirm split the parser under policy B slightly exceeds the 30-question oracle subset (0.833 vs 0.767), which contains two retained oracle construction errors; within sampling

Table 1: **Central attribution results.** Naive observation → attribution control → revised interpretation. Values from `canonical_results.csv` (result IDs in Appendix 9).

Setting	Naive observation	Attribution control	Revised interpretation
TabMWP	Parser 0.850 vs oracle 0.967 (gap 0.117)	Match generation budget (800→1200 tokens)	Parser 0.967 = oracle. The gap was truncation, not representation ability.
TAT-QA (calc.)	Parser 0.683 vs oracle 0.833 (gap 0.150)	Idempotent percent policy (model's ×100 not double-converted)	Parser 0.817, oracle 0.833 (gap 0.016). Most of the naive gap was scorer policy.
MATH (screen)	Parser 0.850 vs oracle 0.983	Crash-isolated executor; failure census	Genuine gap: 6.7% execution-invalid programs, repairable (Sec. 6).
MATH (confirm)	Parser 0.767 vs oracle-subset 0.933	Failure census of the residual	0% execution-invalid; the residual is normalization (10/14) and composition (3/14). Nothing for execution verification to repair.
DROP (numeric)	Parser 0.567 vs direct LM 0.817 (n=60)	Failure census + budget audit	Failures are 800-token truncation on long passages (10/19) and count-instead-of-span choices (7/19).
MATH (clean split)	Repair shows no benefit (all arms 0.767)	Induce the failure mode prospectively (stress budget)	Benefit appears exactly with the induced execution-invalid class ($\Gamma = 0.50$–0.62).

noise at this size, this is further evidence that little representation headroom remains. Answer normalization and scoring policy can change where an end-to-end gap appears to originate. We report both policies; the pattern (most of the gap is policy, not representation) is the finding.

MATH: a genuine repairable class survives. After the executor is crash-isolated and the scorer normalization is canonical, a real representation failure remains: 6.7% of programs are execution-invalid, and the oracle gap is 0.133. Once interface artifacts are controlled, this remaining class can be tested by a targeted intervention, which is exactly what Sections 6–7 do.

Synthesis. Superficially similar end-to-end discrepancies resolved into different explanations once the relevant interface was controlled: budget (TabMWP), scorer policy (TAT-QA), genuine representation failures (MATH screen), evaluation-side residual (MATH confirm), and budget plus answer-type choice (DROP). No single mechanism explains the five rows; that is the point. An accuracy gap is a symptom, and attribution requires intervening on the interfaces one at a time.

A worked example. TAT-QA question `6a9b6186` asks for the increase in OEM net-revenue percentage from 2017 to 2019; the gold answer is 4 (percent). The baseline parser writes `70-66`: correct operands, correct operation, correct value. The pipeline scores it wrong, because the record's scale field is `percent` and the scorer multiplies the result by 100, converting 4 into 400. An end-to-end reading of this case says "the pipeline failed." The attribution reading says: representation construction succeeded, execution succeeded, and the failure entered at the evaluation interface's unit convention. Neither a better representation model nor any amount of sampling fixes this case; a one-line scorer policy does.

5 What baseline failures are made of

We classified every baseline-parser failure on the frozen subsets (61 failures across TAT-QA, MATH, DROP; per-case labels in Appendix I) into: **E1** execution-invalid (parse/schema/crash); **E2** executable but wrong evidence binding; **E3** plausible binding, wrong operation or composition; **E4** evaluation infrastructure (executor policy, normalization, formatting). The census inverts the intuitive picture: E4 is the largest class (24/61), then E3 (17/61), E1 (15/61), E2 (5/61). On MATH confirm, 10 of 14 residual failures are mathematically correct programs whose printed form does

Table 2: **Canonical results.** Accuracy; matched 1200-token budget unless noted. M1 = executable baseline; M2 = sample-and-verify (K=2); M3 = verifier-gated repair ($T \leq 2$); SD = matched self-debugging (generic stderr feedback, same start state and budget). CIs and provenance: Appendix F.

System	Split	Model	M1	Full LM	M2	M3	SD	Oracle-PX
TAT-QA	screen	Kimi	.700	.667	.733	.700	—	.900
TAT-QA	confirm	Kimi	.667	.667	.667	.667	—	.767
TabMWP	screen	Kimi	.967	.950	.967	.950	—	.967
MATH	screen	Kimi	.850	.683	.883	**.933**	.900	.983
MATH	screen	GLM	.733	—	.833	**.933**	.850	.983
MATH	confirm	Kimi	.767	.600	.767	.767	—	.933†
DROP num.	screen	Kimi	.700	.900	—	—	—	.933
DROP num.	confirm	Kimi	.433	.800	—	—	—	.867

† 30-question label-blind oracle subset.

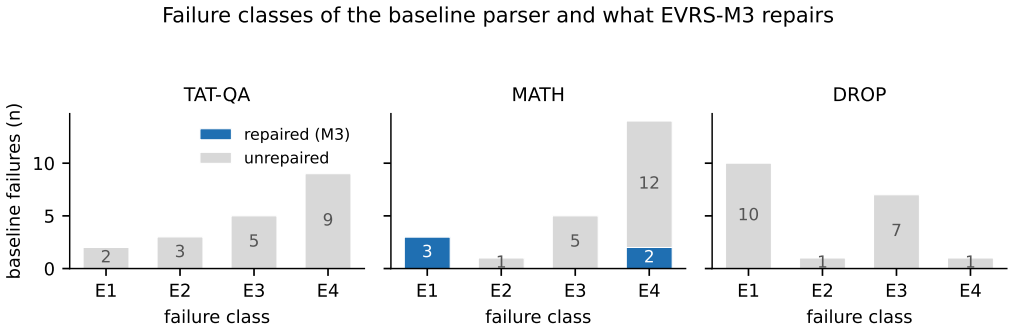

Figure 1: **Failure classes of the baseline parser and what verifier-gated repair fixes.** Repair recovers the E1 class and cannot see E2/E3. E4 failures are evaluation infrastructure, not model errors.

not match the dataset's LaTeX conventions. On TAT-QA, 9 of 19 failures are one scorer policy: the executor converts percent answers by $\times 100$, and models often write `*100` themselves. E2, the class grounded-verification proposals target, is the rarest. On MATH screen specifically, the nine baseline failures split into four formatting (E4), three crashes (E1), and two composition (E3) cases; this composition, not the headline accuracy, predicts that repair gains must be small there (and they were: 3 questions).

Representative cases make the classes concrete. E1: `math-4377` calls `lcm(51,68,85)` with the wrong API convention and crashes; E2: TAT-QA `a0414f81` asks for the 2019 average and the representation averages all three years: every operand is in the table, in an on-topic row, from the wrong column. E3: `math-2046` computes a polygon's signed area as -84 without `abs`; the geometry is right, the orientation is not. E4: `math-2979` prints `Interval(-13, 3)` where the gold is `[-13,3]`; the program is mathematically correct and the scorer cannot see it.

The class structure predicts intervention behavior (Figure 1, Appendix I): execution-feedback repair recovers E1 (3/3 pure crashes on MATH screen) and never fires on E2/E3, whose representations execute successfully by definition. The per-class repair transitions are E1 3/5, E2 0/3, E3 0/5, E4 2/14; the two E1 misses are TAT-QA comparison-operator expressions (e.g., `(94.2>50)+(45.1>50)`) outside the executor's arithmetic whitelist, a capacity limit of the executor rather than of repair. No failure remained unclassified (E5 = 0): every label was assigned against the frozen response, with direct executor re-runs for ambiguous cases.

6 Intervention and Matched Baselines

The intervention serves the attribution hypothesis: if the diagnosis is right, repairing the diagnosed class should change the outcome, and nothing else should change. The intervention is the simplest

Table 3: **Matched interventions and cost** (MATH screen, both models). tok/q = mean output tokens per question; rep./q = repair calls per question. M3 pays repair cost only where verification fails.

Model	Method	Accuracy	tok/q	rep./q
Kimi	M1 (parser)	.850	298	0
Kimi	M2 (sample-verify)	.883	607	0
Kimi	SD (self-debug)	.900	$\sim$330	0.07
Kimi	M3 (verifier-gated)	**.933**	362	0.05
GLM	M1 (parser)	.733	586	0
GLM	M2 (sample-verify)	.833	1108	1.72
GLM	SD (self-debug)	.850	$\sim$700	0.25
GLM	M3 (verifier-gated)	**.933**	885	0.82

one the measurement implies: synthesize one representation, verify it deterministically (schema validity; execution success; provenance: every operand literal must appear verbatim in the evidence), and repair with the verifier's feedback only when verification fails, at most twice (M3). Two properties matter for the causal reading: the failure-class assignment is produced by the verifier on the initial synthesis, before any repair call, and never uses repair outcomes; and the repair loop is deliberately generic, so that a matched baseline can isolate the signal. The verifier logic is shared across datasets; only the schema and executor adapt. Two controls bound the interpretation: M2 (sample-and-verify, $K{=}2$) measures the extra-sample explanation; SD (self-debugging) shares M3's loop and start state but replaces the verifier's targeted feedback with the raw executor error (Chen et al., 2024).

The verifier's four checks contribute unevenly, and the decomposition is itself attribution evidence. Execution success drives the MATH gains: every repaired MATH case was a crash or parse failure. Provenance (literals grounded in source text) carries the table-task value: it is the only check that catches hallucinated operands, invisible to execution; but such cases are rare in these runs (1/120 TAT-QA baseline representations). Schema validity catches JSON format failures; sanity bounds never fired. A provenance *instruction* in the synthesis prompt (asking the model to ground its own literals) was tested and rejected: it cost 10 points on TAT-QA by pushing the model toward span-style answers. Verification belongs in the verifier, not in the prompt.

On MATH screen (Table 2), execution-feedback repair reaches 0.933 on both model families (63% for Kimi and 80% for GLM of the oracle gap; paired difference over the parser $+0.083$ $[+0.017, +0.150]$), against 0.850/0.733 for the parser and 0.683 for direct reasoning. The matched self-debugging baseline recovers nearly the same effect (SD 0.900/0.850; paired M3$-$SD $+0.033$/$+0.083$, intervals touching zero; Sec. 8 interprets this).

Cost: 0.05 repairs per question (Kimi screen), exactly zero on the confirm split; Table 3 shows the accuracy–token dominance over sampling and direct reasoning.

On the table-QA systems the same intervention behaves as the census predicts. TAT-QA screen has a 3% invalid rate and a small gain (M2 0.733 vs parser 0.700); the confirm split ties at 0.667. TabMWP at matched budget has nothing to repair (M1 $=$ oracle $=$ 0.967): the zero-opportunity control. The verifier's provenance channel, unique to the table tasks, catches hallucinated operands that execution alone cannot; such failures are rare in these runs (one flagged case in 120 baseline representations). DROP exposes a different limit: span-family questions score lower through the representation than direct answering (0.433 vs 0.733) because the JSON span schema cannot express the required counting, so DROP remains baselines-only.

7 SCALED PROSPECTIVE ATTRIBUTION

The screen results are retrospective: failures were observed before the intervention was evaluated. The prospective test reverses this order. Each question's representation is diagnosed *before* any repair, using only deployment-time information: the deterministic verifier's verdict on the initial synthesis. Write E_1 for an execution-invalid representation (parse failure, schema violation, or execution crash) and E_0 for a representation without that diagnosed failure. The repair effect on

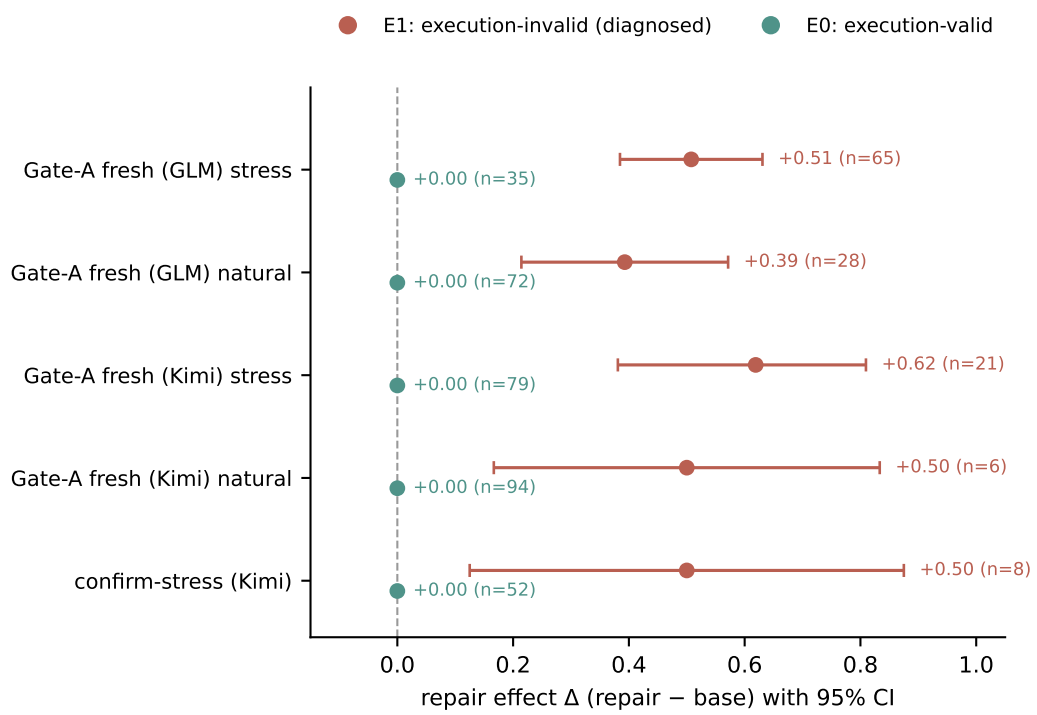

Figure 2: **Attribution predicts intervention response.** Repair effect by diagnosed class (per-question verifier diagnosis before repair) across five prospective settings: the earlier confirm-stress run, the fresh Gate-A sample under natural and stress budgets, and the GLM replication. $\Delta_1 > 0$ throughout; $\Delta_0 = 0$ in 332 execution-valid evaluations; pooled $\Gamma = 0.50$ $[0.40, 0.61]$ (cluster bootstrap by problem).

each class is $\Delta_1 = A_{\text{repair},E_1} - A_{\text{base},E_1}$ and $\Delta_0 = A_{\text{repair},E_0} - A_{\text{base},E_0}$, and the interaction

$$\Gamma = \Delta_1 - \Delta_0$$

measures whether the intervention's benefit is concentrated in the class diagnosed before intervention. This is stronger than a retrospective subgroup analysis: the subgroup labels are produced by a mechanism that runs at inference time, not by post-hoc inspection of errors, so Γ estimates a quantity a deployment could act on: repair exactly what the diagnosis flags.

Protocol (frozen before any model call; Appendix O): draw a fresh level-stratified MATH sample (n=100, disjoint from all prior subsets, seed 20260921); for each question, run the pipeline under a natural budget (1200 tokens) and a stress budget (256 tokens, known to induce truncated, unexecutable programs); diagnose each synthesis with the verifier before repair; then measure the repair effect per class. The two conditions play complementary roles. The natural condition tests the pattern under ordinary inference, establishing that selective intervention opportunity occurs without deliberately inducing failure. The stress condition is a controlled intervention on failure prevalence: it raises the execution-invalid rate and asks whether the same mechanism (repair acts on E1, not on E0) survives the shift.

Figure 2 shows the result; Table 4 gives the numbers.

On the fresh sample (Kimi), stress induces a 21% invalid rate; repair recovers $\Delta_1 = 0.62$ on the diagnosed-invalid questions and $\Delta_0 = 0.00$ on the unflagged rest, $\Gamma = 0.62$ $[0.40, 0.82]$. The natural budget on the same questions shows the same signature at lower density (invalid rate 6%, $\Gamma = 0.50$); Pooled across all prospective settings (the fresh sample under both budgets, the earlier confirm-stress run, and the cross-model replication below), the interaction is

$$\Gamma_{\text{pooled}} = 0.500, \qquad 95\%\ \text{CI}\ [0.397, 0.609],$$

Table 4: **Scaled prospective attribution** (frozen protocol; fresh n=100 MATH sample, disjoint from all prior subsets; E0/E1 diagnosed by the verifier before repair). The earlier confirm-stress run (n=60) replicates the same pattern (Δ_1=0.50, Δ_0=0.00). The pooled row's 460 evaluations span 160 unique problems: 100 gate problems under both budgets and both model families, plus 60 confirm-stress problems — all pre-registered prospective runs.

Model	Condition	invalid rate	M1	M3	Δ_1	Δ_0	Γ [95% CI]
Kimi	natural	6%	.770	.800	+0.50	0.00	0.50 [0.00, 1.00]
Kimi	stress	21%	.680	.810	+0.62	0.00	0.62 [0.40, 0.82]
GLM	natural	28%	.660	.770	+0.39	0.00	0.39 [0.21, 0.58]
GLM	stress	65%	.340	.670	+0.51	0.00	0.51 [0.39, 0.63]
Kimi	confirm-stress (n=60)	13%	.717	.783	+0.50	0.00	0.50 [0.14, 0.88]
pooled (460 evaluations, 160 problems)					+0.50	0.00	0.50 [0.40, 0.61]

over 460 question-condition evaluations spanning 160 unique problems (cluster bootstrap by problem). $\Delta_1 > 0$ in every condition: the diagnosis identifies a class with substantial recoverable error. $\Delta_0 = 0$ in every condition; here E_0 is the unflagged class under the gated policy — those representations receive no repair — so $\Delta_0 = 0$ records the gate's precision, not a treatment effect on valid representations. On the paired level, repair helped 64 of 460 evaluations and harmed none.

The GLM replication on identical questions shows a higher invalid rate (28% natural, 65% stress, vs 6%/21% for Kimi) and the same selectivity ($\Gamma = 0.39\ [0.21, 0.58]$ natural; $0.51\ [0.39, 0.63]$ stress). The model-specific effect sizes differ (the protocol's frozen magnitude prediction, GLM $\Delta_1 \geq$ Kimi Δ_1, failed: GLM 0.51 versus Kimi 0.62 under stress), while the direction and the E1-confinement replicate. We treat this as ordinary effect-size heterogeneity: how much repair recovers depends on the model's failure distribution, but where it acts does not.

An earlier run of the same design on the held-out MATH confirm split (n=60) replicates the pattern and supplies the pre-registration detail. Four predictions were logged before that run: the 256-token budget would induce an invalid rate of at least 10% (observed 13.3%); repair would recover at least 40% of the induced gap (observed 133%); verifier-targeted feedback would beat generic error feedback (failed — matched self-debugging reached the identical 0.783); repair cost would stay under 0.6 calls per question (observed 0.18). The failed prediction is the informative one: on crash-type failures the traceback largely speaks for itself (Sec. 8).

The natural-budget confirm arm supplies the complementary observation: 0% invalid, all methods tie at 0.767, no repair fires. The frozen decision rules (Appendix O) were met at Gate A; the breadth gate (GLM) confirmed cross-model. Attribution, made before intervention by a deterministic check, predicts where the intervention acts.

8 Boundaries of execution-feedback repair

The effect belongs to the intervention class, not the architecture. Matched self-debugging (the same loop, start state, and budget with raw executor errors as feedback (Chen et al., 2024)) reproduces the repair effect: an exact tie under the stress condition (0/60 per-question disagreements) and differences of +0.033/+0.083 with intervals touching zero on screen. Execution feedback, not verifier-specific machinery, is the operative signal for the diagnosed class. The verifier's distinct contributions are the deterministic gate that spends repair only where it fires and the provenance channel on table tasks.

Execution-valid semantic errors are a different class. E2/E3 representations execute, pass provenance, and still answer the wrong question; execution verification has zero recall on them by construction (all 22 frozen E2/E3 failures pass every check). A deterministic grounding diagnostic (provenance rate, temporal-header match, row-label overlap, operand-coverage counts) separates E2 from correct cases with recall 1/3 and frequent false alarms on correct ones (mean provenance 0.58 on E2 vs 0.81 on correct, with overlapping ranges); the wrong operands are typically in-source, in-year, and on-topic, so surface grounding cannot see the error.

Semantic-aware correction recovers some, not most. An adaptation of EVER's self-guided semantic correction (Zhang et al., 2026) (the model inspects its representation and the execution result, then confirms or corrects, one round, matched budget) recovers 3/22 frozen E2/E3 failures (13.6%) at a 3.3% false-repair cost on 30 correct controls, and usually confirms its own wrong representation rather than correcting it (Appendix L). Semantic-aware verification can therefore repair part of the residual class; most of it remains unresolved by the mechanisms tested here.

Prevalence is not repairability, and clean splits stay clean. GLM produces execution-invalid representations far more often than Kimi (65% vs 21% under stress) yet recovers a smaller share per case (Δ_1 0.51 vs 0.62): failure prevalence predicts repair opportunity, not per-case repairability. And on splits where the diagnosis finds nothing (MATH confirm: 0% invalid, all arms tie at 0.767; TabMWP at matched budget: nothing to repair), the intervention is inert, as the attribution account requires.

9 Discussion and conclusion

A practical sequence. For evaluating an executable-reasoning pipeline (each step demonstrated on a real gap in this paper): (1) match the generation budget before comparing representations; (2) inspect representation construction with a substitution control before tuning the model; (3) control the deterministic executor (crash isolation, timeout behavior under load); (4) audit normalization and scoring conventions before interpreting a gap; (5) test proposed mechanisms with targeted interventions, prospectively.

What changes for evaluation practice. The attribution strategy applies directly to hybrid LM/executor systems with observable intermediate representations and independently executable computation. For program-aided reasoning, a method comparison that holds the model fixed but not the budget or scorer measures the harness, not the method; both TabMWP and TAT-QA changed conclusions under such controls. For tool-augmented models, the same logic applies to tool-call arguments: a wrong-but-executable call is the tool-use analog of our E2/E3 class, and our results predict it will dominate once crash-type failures are engineered away. For self-correction mechanisms, the prospective interaction gives a deployment rule: run a cheap execution-validity check first, and spend repair compute only where it fires; in our data this is exactly as good as repairing everywhere and never harmful. The core implication is that a higher end-to-end score does not by itself identify which computational component improved; attribution requires an intervention on the interface one suspects.

Scope. The attribution studied here predicts response to execution feedback for the diagnosed execution-invalid class, in quantitative reasoning with deterministic executors, on two model families; other failure classes require evidence channels that expose semantic or task-level correctness. The prospective gates ran on MATH; the mechanism's prediction for table QA follows from the TAT-QA screen result (a small gain at a 3% invalid rate) but was not gate-tested there. Subsets are small (30–100 per split) by design: every number is cache-audited and regenerable, and the evidence for the mechanism is the replication pattern across five prospective settings, not any single split. The claim survives its own falsification conditions: no repair of execution-valid representations (332 evaluations), cross-model replication, and no gains on zero-opportunity splits. The debunked gaps are our own pipeline's naive readings, presented as illustrations of a measurement failure mode that is easy to commit; we do not claim any specific published number is wrong.

Conclusion. Executable language-model reasoning fails in specific, separable places. Controlled substitutions show that two apparent representation gaps are budget and scorer artifacts; that genuine execution-invalid failures are recovered by execution feedback exactly where they are diagnosed; and that executable-but-semantically-wrong representations remain outside both execution validity and current semantic-correction methods. Attribution before intervention.

REPRODUCIBILITY STATEMENT

(This section is **recommended** and does not count toward the page limit.)

The reproducibility map links each main figure, table, and claim to its result IDs, configurations, analysis scripts, and canonical evaluation artifacts. The controlled method comparisons, prospective gate results, matched self-debugging control, canonical result tables, and per-case failure labels are recorded with the corresponding experimental conditions.

The supplementary material contains the canonical evaluation data, experiment configurations, per-case failure labels, intervention results, statistical outputs, and analysis code used for the reported experiments. The supplied scripts recompute the main attribution, intervention, failure-rate, and statistical analyses and regenerate the corresponding tables from the released result records.

AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

In this work, we used generative AI tools for providing feedback on research methodology and experiments, providing feedback on mathematical arguments, and interpreting results. We have not used generative AI tools for generating synthetic data sets, formulating mathematical claims or implementing methods, and the remaining required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for literature discovery, brainstorming, drafting and editing parts of the manuscript, editing software code and figures, and simulated peer-review feedback. We have reviewed all AI-assisted work and AI-assisted code was re-run with its outputs checked against the experimental artifacts reported in this paper. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *TMLR*, 2023.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *ICLR*, 2024.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *NAACL*, 2019.

Jiazhan Feng, Shijue Huang, Xingchen Xing, et al. Retool: Reinforcement learning for strategic tool use in LLMs. *arXiv:2504.11536*, 2025.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, 2023.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Min Yang, Nan Duan, and Weizhu Chen. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *ICLR*, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *NeurIPS Datasets and Benchmarks*, 2021.

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

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. Tabmwp: Tabular math word problems. In *ICLR*, 2023.

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

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Minghan Zhang, Zhen Yang, Haodong Zou, Jie Chen, Zhen Duan, and Shu Zhao. Execution as verification: Fine-grained self-correcting reasoning for complex KBQA. In *ACL*, 2026.

Yuchen Zhou et al. Reveal: Self-evolving code agents via reliable self-verification. In *ICLR*, 2026.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. In *ACL*, 2021.

A FORMAL SETUP

Pipeline $\hat{a} = S(E(z))$, $z = g_\theta(q, e)$. Arms: A_L (direct), A_{PX} (pipeline), A_{OPX} (oracle representation, same E/S), A_0 (no evidence). Quantities: $V_P = A_{\mathrm{OPX}} - A_{\mathrm{PX}}$; $V_R = A_L - A_{\mathrm{PX}}$; $R_P = (A_{\mathrm{method}} - A_{\mathrm{PX}})/V_P$; stress recovery $R_{\mathrm{stress}} = (A_{\mathrm{evrs}} - A_{\mathrm{stressed}})/(A_{\mathrm{base}} - A_{\mathrm{stressed}})$; per-class effects $\Delta_1 = \mathbb{E}[\mathrm{acc}_{\mathrm{M3}} - \mathrm{acc}_{\mathrm{M1}} \mid E_1]$, Δ_0 on E_0; interaction $\Gamma = \Delta_1 - \Delta_0$; pooled Γ aggregates all prospective settings at the question level. All intervals are per-question paired bootstrap (10,000 resamples, seed 7); per-arm intervals resample questions within an arm. The E0/E1 assignment is the deterministic verifier verdict on the initial synthesis, computed before any repair call.

B DATASET AND SUBSET CONSTRUCTION

Sources: TAT-QA (Zhu et al., 2021) dev split (calculable question family only; extraction-family questions are excluded because their answers are spans, not executor outputs); TabMWP (Lu et al., 2023) test problems; MATH (Hendrycks et al., 2021) test problems from the ToRA distribution, level-stratified; DROP (Dua et al., 2019) dev split (numeric-answer questions for the executable subsets; span-family questions appear only in the baselines-only analysis). Table 6 (below, with the other generated tables) lists every subset with its sample-id hash. The MATH gate samples are level-stratified and disjoint from screen/confirm by construction (the sampler removes the 120 previously used problems before drawing). The DROP baseline arm used an 800-token generation budget; all other reported arms used 1200 tokens.

C MODEL AND INFERENCE CONFIGURATIONS

Kimi-K3 and GLM-5.3-Flash via an API gateway; default temperature; thinking disabled; max output tokens 1200 (standard), 256 (stress), 800 (DROP budget). Retry policy: one regeneration on empty or unparseable output; the failure is kept as data otherwise. Cache key: prompt SHA-256 + model tag + arm; cache reads verify the prompt hash, and validators reject empty/unparseable responses at write time.

D CANONICAL EXECUTOR

Two executors, both deterministic. `safe_arith` (JSON expressions): an AST whitelist over $+, -, \times, \div$, parentheses, `min/max/abs`; thousand-separator commas are stripped; a single-element list wrapping is unwrapped; anything else is a rejected (E1) representation. `exec_program` (MATH): the sympy program runs in a subprocess with the interpreter's site-packages on PYTHONPATH, a hard timeout, and a per-process result cache; the subprocess isolates the parent from C-level crashes. Unit tests cover: identity/no-op cases; both percent policies; comma literals; list wrapping; whitelist rejection of comparisons; subprocess timeout (verified to trigger at 3 s); subprocess survival of a sympy crash; normalization identity cases for each supported LaTeX form.

E CANONICAL SCORER AND NORMALIZATION

TAT-QA: the official EM/F1 metric with the record's scale field; the metric rounds to two decimals, verified directly. Policy A (canonical, frozen before measurement): percent-scale answers multiplied by 100 after expression evaluation. Policy B (sensitivity): idempotent — skip the conversion when the expression already contains an explicit 100-factor or contains no division. All nine TAT-QA percent-policy failure cases were re-executed by hand under both policies; all nine are correct values under policy B. MATH: normalizer v2 handles LaTeX fractions (including nested), radicals, degrees, implicit multiplication, tuples, `pmatrix`, and the imaginary unit; known residual gaps (Interval/Union printing, term order in complex sums) are classified E4 rather than papered over. TabMWP: comma-tolerant numeric comparison, relative tolerance 10^{-3}. DROP: official EM/F1.

F FULL CANONICAL ATTRIBUTION TABLES

Every arm of every setting, with bootstrap intervals, invalid-representation rates, execution-success rates, token means, call counts, sample-id hashes, and scorer/executor versions: `canonical_results.csv` (68 rows). The three tables below are generated from it directly.

Table 5: Full canonical attribution table. One row per (system, split, model, arm). inv = baseline invalid-representation rate; exec = execution-success rate; tok = mean output tokens per question. CIs are 95% bootstrap over questions.

system	split	model	arm	n	acc	95% CI	inv	tok
TAT-QA	screen	kimi	APX (M1)	30	.700	[.533, .867]	.033	203
TAT-QA	screen	kimi	EVRS-M2	30	.733	[.567, .867]	—	399
TAT-QA	screen	kimi	EVRS-M3	30	.700	[.533, .867]	—	214
TAT-QA	screen	kimi	EVRS-M4	30	.733	[.567, .867]	—	399
TAT-QA	screen	kimi	Full LLM (CoT)	30	.667	[.500, .833]	—	—
TAT-QA	screen	kimi	Oracle-PX	30	.900	[.800, 1.000]	—	—
TAT-QA	confirm	kimi	APX (M1)	30	.667	[.500, .833]	.033	153
TAT-QA	confirm	kimi	EVRS-M2	30	.667	[.500, .833]	—	321
TAT-QA	confirm	kimi	EVRS-M3	30	.667	[.500, .833]	—	163
TAT-QA	confirm	kimi	EVRS-M4	30	.667	[.500, .833]	—	324
TAT-QA	confirm	kimi	Full LLM (CoT)	30	.667	[.500, .833]	—	—
TAT-QA	confirm	kimi	Oracle-PX	30	.767	[.600, .900]	—	—
TabMWP	screen	kimi	APX (M1)	60	.967	[.917, 1.000]	0.000	208
TabMWP	screen	kimi	EVRS-M2	60	.967	[.917, 1.000]	—	338
TabMWP	screen	kimi	EVRS-M3	60	.950	[.883, 1.000]	—	318
TabMWP	screen	kimi	EVRS-M4	60	.933	[.867, .983]	—	405
TabMWP	screen	kimi	Full LLM (CoT)	60	.950	[.883, 1.000]	—	—
TabMWP	screen	kimi	Oracle-PX	60	.967	[.917, 1.000]	—	—
MATH	screen	kimi	APX (M1)	60	.850	[.750, .933]	.067	298
MATH	screen	kimi	EVRS-M2	60	.883	[.800, .950]	—	607
MATH	screen	kimi	EVRS-M3	60	.933	[.867, .983]	—	362
MATH	screen	kimi	EVRS-M4	60	.917	[.833, .983]	—	627
MATH	screen	glm-5.3-flash	APX (M1)	60	.733	[.617, .833]	.200	586
MATH	screen	glm-5.3-flash	EVRS-M2	60	.833	[.733, .917]	—	1108
MATH	screen	glm-5.3-flash	EVRS-M3	60	.933	[.867, .983]	—	885
MATH	screen	glm-5.3-flash	EVRS-M4	60	.883	[.800, .950]	—	1229
MATH	screen	kimi	Full LLM (CoT)	60	.683	[.567, .800]	—	—
MATH	screen	kimi	Oracle-PX	59	.983	[.949, 1.000]	—	—
MATH	screen	kimi	Self-Debug (generic stderr)	60	.900	[.817, .967]	—	—
MATH	screen	glm-5.3-flash	Self-Debug (generic stderr)	60	.850	[.766, .933]	—	—
MATH	confirm	kimi	APX (M1)	60	.767	[.650, .867]	0.000	315
MATH	confirm	kimi	EVRS-M2	60	.767	[.650, .867]	—	646
MATH	confirm	kimi	EVRS-M3	60	.767	[.650, .867]	—	315
MATH	confirm	kimi	EVRS-M4	60	.767	[.650, .867]	—	646
MATH	confirm	kimi	Full LLM (CoT)	60	.600	[.467, .717]	—	—
DROP	screen	kimi	APX (M1)	30	.700	[.533, .867]	—	—
DROP	screen	kimi	Full LLM (CoT)	30	.900	[.767, 1.000]	—	—
DROP	screen	kimi	Oracle-PX	30	.933	[.833, 1.000]	—	—
DROP	confirm	kimi	APX (M1)	30	.433	[.267, .600]	—	—
DROP	confirm	kimi	Full LLM (CoT)	30	.800	[.667, .933]	—	—
DROP	confirm	kimi	Oracle-PX	30	.867	[.733, .967]	—	—
MATH	confirm-stress256	kimi	APX (M1)	60	.717	[.600, .833]	.133	—
MATH	confirm-stress256	kimi	EVRS-M3	60	.783	[.667, .883]	—	—
MATH	confirm-stress256	kimi	Self-Debug (generic stderr)	60	.783	[.667, .883]	—	—
TAT-QA	screen	kimi	APX (M1) [policy B]	30	.800	—	—	—
TAT-QA	screen	kimi	EVRS-M2 [policy B]	30	.833	—	—	—
TAT-QA	screen	kimi	EVRS-M3 [policy B]	30	.800	—	—	—
TAT-QA	screen	kimi	Oracle-PX [policy B]	30	.900	—	—	—
TAT-QA	confirm	kimi	APX (M1) [policy B]	30	.833	—	—	—
TAT-QA	confirm	kimi	EVRS-M2 [policy B]	30	.833	—	—	—
TAT-QA	confirm	kimi	EVRS-M3 [policy B]	30	.833	—	—	—
TAT-QA	confirm	kimi	Oracle-PX [policy B]	30	.767	—	—	—

system	split	model	arm	n	acc	95% CI	inv	tok
MULTI	frozen-e23	kimi	EVER-style semantic correction	22	.136	[0.000, .273]	—	—
MULTI	frozen-controls	kimi	EVER-style semantic correction	30	.033	[0.000, .100]	—	—
CCPP	round1-frozen	kimi	Direct answer	160	.544	—	—	—
CCPP	round1-frozen	kimi	Self-Consistency N=8	160	.637	—	—	—
CCPP	round1-frozen	kimi	Hidden verification	160	.812	—	—	—
CCPP	round1-frozen	kimi	Visible feedback	160	.956	—	—	—
CCPP	round1-frozen	kimi	Direct physical decision	160	.950	—	—	—
MATH	gateA-nat	kimi	APX (M1)	100	.770	[.690, .850]	.060	—
MATH	gateA-nat	kimi	EVRS-M3	100	.800	[.720, .880]	—	—
MATH	gateA-str	kimi	APX (M1)	100	.680	[.590, .770]	.210	—
MATH	gateA-str	kimi	EVRS-M3	100	.810	[.730, .880]	—	—
MATH	gateA-nat	glm-5.3-flash	APX (M1)	100	.660	[.570, .750]	.280	—
MATH	gateA-nat	glm-5.3-flash	EVRS-M3	100	.770	[.690, .850]	—	—
MATH	gateA-str	glm-5.3-flash	APX (M1)	100	.340	[.250, .440]	.650	—
MATH	gateA-str	glm-5.3-flash	EVRS-M3	100	.670	[.570, .760]	—	—
MATH	pooled-prospective	kimi+glm	EVRS-M3 vs base	460	—	—	—	—

Table 6: Frozen subsets. Hashes are the sample-id set hashes; all subsets were drawn before measurement with fixed seeds.

dataset	subset	n	hash	answer type
TAT-QA (dev)	calculable screen	30	`a8f5c3208e08`	numeric
TAT-QA (dev)	calculable confirm	30	`147f94e0c749`	numeric
TabMWP	screen	60	`e6f44f341876`	numeric
MATH	screen	60	`b129b26d6613`	boxed LaTeX
MATH	confirm	60	`306b2a8e683b`	boxed LaTeX
MATH	gate-A (fresh)	100	seed 20260921	boxed LaTeX
MATH	gate-B reserve (unused)	100	seed 20260921	boxed LaTeX
DROP (dev)	numeric screen	30	`5ff5e51a3211`	numeric
DROP (dev)	numeric confirm	30	`7584e6d8f095`	numeric

Table 7: Cross-model evidence on MATH (identical questions). inv = baseline invalid-representation rate; Δ_1 = repair effect on execution-invalid cases.

model	setting	inv. rate	M1	M3	Δ_1	tok/q (M1)
Kimi	screen	6.7%	.850	.933	—	298
GLM	screen	20.0%	.733	.933	—	586
Kimi	gate natural	6%	.770	.800	+0.50	—
Kimi	gate stress	21%	.680	.810	+0.62	—
GLM	gate natural	28%	.660	.770	+0.39	—
GLM	gate stress	65%	.340	.670	+0.51	—

G GENERATION-BUDGET CONTROLS

Truncation prevalence at the archived 800-token budget: 14/60 TabMWP responses empty (silently truncated); the matched 1200-token budget with one regeneration removes them, and the parser's accuracy moves 0.850 → 0.967. TAT-QA moves 0.683 → 0.700/0.667 under the same control. The 256-token stressor raises the MATH invalid rate from 0% (confirm) to 13.3%, and on the gate-A sample from 6% to 21% (Kimi) and 28% to 65% (GLM); GLM's larger increase reflects its longer programs.

H SCORING-POLICY SENSITIVITY

TAT-QA, both policies, all arms (calculable sets): policy A / policy B — screen: M1 0.700/0.800, M2 0.733/0.833, M3 0.700/0.800, oracle 0.900/0.900; confirm: M1 0.667/0.833, M2 0.667/0.833,

M3 0.667/0.833, oracle 0.767/0.767. The oracle's invariance is structural: label-blind oracle derivations are ratio-style and convert exactly once under either policy. MATH: the audit-stage $V_R = +0.183$ was largely a normalization artifact; under normalizer v2 the same outputs score $A_L = A_{\mathrm{PX}} = 0.642$ on the audit subset.

I FAILURE TAXONOMY

Definitions (main text §5). Procedure: every baseline failure on the frozen subsets (61 cases) was labeled against its frozen cached response; ambiguous cases were resolved by re-executing the representation with the canonical executor and comparing against the gold with the canonical scorer. No case remained unresolved (E5 = 0). Counts: E1 15 (TAT-QA 2, MATH 3, DROP 10), E2 5, E3 17, E4 24.

Per-class canonical examples (uid; representation; diagnosis):

- **E1** `math-4377`: `lcm(51,68,85)` — argument-convention misuse; crashes. `math-3410`: `sum(digits(n, base))` — the call returns a structure, not a list; crashes. TAT-QA `3d384cee`: `(94.2>50)+...` — comparison operators are outside the arithmetic whitelist.
- **E2** TAT-QA `a0414f81`: `(166+178+192)/3` for a 2019-specific question — all three operands are in the table, in the right row, from the wrong columns. TAT-QA `806fd692`: `4422+15400` — a liability component is missing from the sum.
- **E3** TAT-QA `524463b6`: `42056-9982` where the gold is -32074 — right values, reversed order. `math-2046`: `Polygon(...).area` returns -84 — correct geometry, signed orientation. `math-1368`: `factorial(2)` for a three-digit counting problem — the combinatorial structure is absent.
- **E4** TAT-QA `6a9b6186`: `70-66` $= 4$, correct, scored 400 (percent double-conversion). `math-2979`: prints `Interval(-13, 3)` against gold `[-13,3]`. `math-4813`: floats against gold fractions. `math-3881`: `4:30` against gold `\text{4:30 p.m.}`. `math-4706`: a correct program that failed only under parallel executor load (timeout flake; correct in isolation).

Repair transitions (M3 vs M1, per class): E1 3/5, E2 0/3, E3 0/5, E4 2/14. The two E4 recoveries co-occur with executor-visible failures; execution feedback never repaired a purely semantic or purely evaluative case.

J REPAIR INTERVENTION

M3 (verifier-gated repair): synthesize once; verify (schema, execution, provenance, sanity); on failure, repair with the verifier's feedback string; at most T=2 rounds. MATH repair prompt: *"Your previous program failed verification. Feedback: {EXEC_ERROR|PARSE_ERROR|...}. Previous program: ... Return ONLY the corrected complete program (self-contained, ends with print(answer)). Problem: ..."*. Table-task feedback additionally names the failed check and the failing literals. M2 samples K=2 and takes the first candidate passing verification. M4 (verify-then-repair over sampled candidates) adds nothing over M2 (0.917 vs 0.883 on MATH screen). Native-cost and matched-budget numbers: Table 3 and `canonical_results.csv`.

K SELF-DEBUG COMPARISON

SD shares M3's loop, start state (the identical cached synthesis response), and budget; the feedback is the raw executor stderr tail. Prompt: *"Your previous program failed to produce an answer. Error output: {stderr tail}. Previous program: ... Return ONLY the corrected complete program ..."*. MATH screen: SD 0.900 (Kimi) / 0.850 (GLM) vs M3 0.933/0.933; paired M3−SD $+0.033$ $[0, +0.083]$ / $+0.083$ $[0, +0.167]$. Stress condition: exact tie (0.783 both), 0/60 per-question disagreements. Reading: the loop and the traceback carry the recovery on crash-type failures; the verifier's deterministic gate decides when to spend it, and its provenance channel is the only part a raw traceback lacks.

L EVER / SEMANTIC-CORRECTION TEST

Adaptation contract (inference-time; EVER has no released code): the model receives the question, the full evidence, its frozen representation, and the execution result; it answers three explicit questions (operands from the asked evidence regions? operation matches the question? evidence missing or extra?) and outputs CONFIRM or one corrected representation in the same schema; one round, matched 1200-token budget. Cases: all 22 frozen E2/E3 failures; 30 frozen correct controls. Outcomes: recovery 3/22 (13.6%); false repairs 1/30 (3.3%); the model attempted changes on 31.8% of failures and otherwise confirmed its own wrong representation. The three recoveries: a missing-addend case (TAT-QA `806fd692`), a combinatorial-structure case (`math-1368`), and a count-instead-of-span case (DROP `bc21502d`). Fidelity limits: single round where EVER iterates; our domain lacks KB structural signals (relation types, empty-result detection), so 13.6% is a lower bound for richer-structure domains and the right estimate for this one. Gate criteria for opening a new method (set beforehand): prior work fails systematically AND a discriminating deterministic signal exists AND a minimal held-out intervention is feasible; the second fails; development stopped. Decision record: RESIDUAL UNSOLVED.

M ROUND-1/ROUND-2 CONTROLS

Round-2's Dispatch Desk (a synthetic dispatch benchmark with verifiable rules) tested whether genuine complementarity exists between predictor state and language reasoning: a programmatic parser→rule→executor pipeline (0.896) tied the full-information LLM (0.896; paired difference 0.000 $[-0.104, +0.104]$). When the rule can be extracted, execution is free and the LLM's residual contribution is nil — the observation that motivated the representation-centric framing.

N PROSPECTIVE E7 EXPERIMENT

Frozen predictions (logged before the run): on MATH confirm (0% invalid at 1200 tokens), compress synthesis to 256 tokens; (i) invalid rate $\geq 10\%$; (ii) $R_{\text{stress}} \geq 0.4$; (iii) M3 $>$ SD; (iv) repair cost ≤ 0.6 calls per question. Results: invalid rate 13.3% (8/60); A_{stressed} 0.717; A_{evrs} 0.783 ($R_{\text{stress}} = 1.33$; paired EVRS$-$stressed $+0.067$ $[+0.017, +0.133]$); A_{sd} 0.783 (prediction (iii) fails; 0/60 per-question disagreements); repairs 0.18 (EVRS) / 0.15 (SD) calls per question. Total cost: $\sim$80 model calls.

O SCALED PROSPECTIVE ATTRIBUTION (GATE PROTOCOL)

Frozen protocol (before any new model call): fresh level-stratified MATH sample n=100, disjoint from all prior subsets, seed 20260921, ids frozen in `gateA_ids.json`; conditions natural (1200 tokens) and stress (256 tokens); per-question E0/E1 diagnosis by the deterministic verifier before repair; quantities $\Delta_1, \Delta_0, \Gamma$. Frozen decision rules: proceed past Gate A iff invalid rate $\geq 8\%$, $\Delta_1 \geq 0.25$, $\Delta_0 \leq 0.05$, $\Gamma > 0$; skip Gate B if the Γ interval excludes zero; Gate C (breadth) uses GLM-5.3-Flash on the same questions with the frozen directional prediction that its invalid rate exceeds Kimi's. Results: Table 4 (main text) and Table 7. Gate A passed all four conditions; Gate B was skipped per the early-stop rule (its reserve sample, 100 further frozen questions, remains unused); Gate C confirmed selectivity cross-model, with the magnitude prediction failing (GLM Δ_1 0.51 $<$ Kimi 0.62 under stress). Per-question outcomes: `fast_scaleup/results/gateA*.csv`.

P CROSS-MODEL EVIDENCE

Table 7 (in the generated tables block) gives the per-model numbers. GLM fails at representation construction more often under every condition (invalid rates 20% vs 6.7% on screen; 28%/65% vs 6%/21% on the gates) and gains more in absolute accuracy from repair (M3$-$M1 $+20$ vs $+8.3$ points on screen), while its per-invalid-case recovery is lower (Δ_1 0.39–0.51 vs 0.50–0.62). GLM is more verbose (886–1229 output tokens per question vs 298–646). Accuracy orderings replicate; cost points do not.

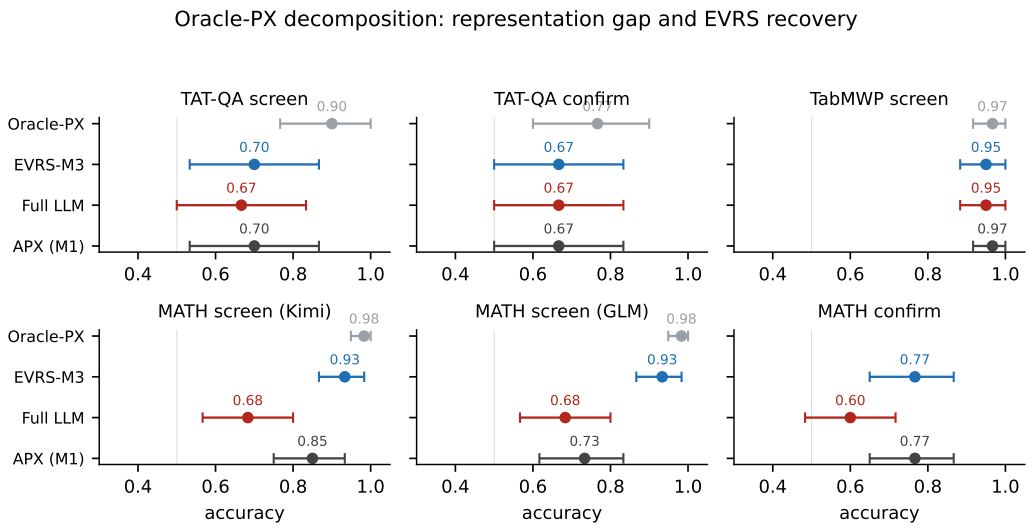

Figure 3: **Oracle-PX decomposition with 95% bootstrap intervals.** Where the parser already sits at the oracle (TabMWP) or the split has no invalid representations (MATH confirm), there is nothing to recover.

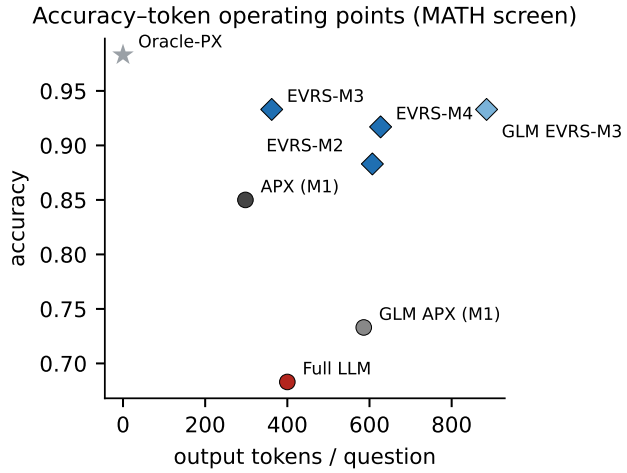

Figure 4: **Accuracy versus output tokens per question** (MATH screen). Verifier-gated repair dominates sampling and direct reasoning on the accuracy–token plane; GLM replicates the ordering.

Q Cost accounting

Per-arm token/call means are columns of `canonical_results.csv`. Repair-vs-sampling efficiency: repair spends 0.05 extra calls per question where M2 spends 1.0; at the screen-set recovery counts this is $\sim 8\times$ fewer extra tokens per recovered question. Selective-deployment cost at the observed natural invalid rates: repair fires on 6% (Kimi) of questions, i.e., the method's overhead is 0.09 calls per question at natural budget.

R Additional plots

Figure 3 (per-system oracle decomposition with bootstrap intervals); Figure 4 (accuracy–token operating points, MATH screen, both models).

S PROMPT TEMPLATES AND SCHEMAS

TAT-QA/TabMWP synthesis prompt (verbatim, frozen v4): *"You are a semantic parser. Read the financial report excerpt and the question. Decompose the question into a deterministic computation over values you extract, and return ONLY a JSON object:* {`"answer_type": "arithmetic"|"count"|"span"|"multi-span", "expression": "<pure arithmetic over extracted numbers ...>", "spans": ["<exact text span(s) ...>"]`} *Rules: numbers in the expression must come from the excerpt verbatim (convert formats like $1,452.4 to 1452.4); no variables, no units, no text inside the expression; spans must be exact substrings of the excerpt."* followed by the context and question. MATH synthesis prompt (verbatim): *"Write a complete Python program using sympy that solves the following math problem and prints ONLY the final answer (no explanation, no labels, just the value/expression). The program must be self-contained and end with print(answer). Problem: ..."*. Repair and SD prompts: Appendices J–K. EVER-adaptation prompt: Appendix L. Stress condition: identical prompts at a 256-token budget. A provenance-instruction variant of the synthesis prompt was tested and rejected (-10 points on TAT-QA).

T REPRESENTATIVE EXAMPLES

Six structured cases (uids resolve to per-case records): correctly attributed execution failure — `math-4377` (`lcm` API misuse; crash; repaired by M3). Repair success under stress — the 13 repaired gate-A stress cases. Repair failure — TAT-QA `3d384cee` (whitelist limit; repair produced no supported-ops rewrite). Executable semantic error — TAT-QA `a0414f81` (all-years average; verifier passes; never repaired). Scoring/normalization example — `math-2979` (correct program, `Interval` printing). Zero-opportunity example — MATH confirm (0% invalid; all arms 0.767). The set deliberately includes failures of the intervention, not only successes.

U SANITY CHECKS

Executor determinism: re-executing all frozen MATH programs twice gives identical outputs. Cache integrity: prompt-hash check on read; a 6-file malformed-cache-entry incident during regeneration (empty model name) was detected by mtime audit, deleted, and re-run correctly. Scorer identity: the official TAT-QA metric's two-decimal rounding was verified directly on the nine percent-policy failure cases. Oracle construction audit: two construction errors retained (documented), counting against the oracle. Baseline regeneration: the full TAT-QA and TabMWP baseline columns were recomputed from cache with zero new calls and matched the recorded summaries after the canonical-scorer regeneration.