BEYOND TOOL–RAW COMPARISONS: MATCHED-CONTROL ATTRIBUTION OF SEMANTIC ASSISTANCE FOR CODE REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-versus-Raw evaluations show whether semantic assistance helps, but they mix useful content with prompt surface and computation. We develop source-verified matched-control attribution and apply it to 398 Python and Bash families in 6,608 calls. Source-only SemCheck and TraceCheck improve exact execution prediction by 12.2 and 15.9 percentage points over truthful but irrelevant notes. The gains persist under exact provider-token matching and when both arms use identical neutral labels (+10.6 points in Python and +12.0 in Bash). On a separate 44-stage resolved-state stress test for Claude Opus 4.8, GPT-5.6 Sol, and Gemini 3.8 Flash, an exact late-stage snapshot raises accuracy from 38.9% to 74.4% (+35.6 points). Across matched tool-reliance analyses, Claude Opus 4.8 and GPT-5.6 Sol solve 92.5% of the programs unaided, yet a sparse causal error reduces accuracy to 0.0% and all 40 responses reproduce the tool-induced wrong answer; a verification instruction restores 97.5% accuracy without adding input tokens. Same-bank controls detect no average advantage from source localization, rule filtering, or a full trace over sufficient plain state: the full-trace interval bounds any positive gain to +1.0 point, while localization remains less precisely resolved at up to +7.2 points. The central result is actionable: applicable rules and resolved state drive improvement, simpler delivery retains the gains, and models that can solve a program unaided may still propagate an erroneous tool state exactly.

1 INTRODUCTION

Code models can generate programs that pass tests while still failing to predict what short programs do. Execution benchmarks such as CRUXEval make this gap directly measurable (Gu et al., 2024), while recent work localizes semantic failures in real projects and across languages (Roy et al., 2026; Li et al., 2026). These failures are especially revealing when behavior turns on implementation semantics—descriptor precedence in Python, process boundaries in Bash, or another rule whose consequence must be carried through a composed state transition.

Rules, traces, and static-analysis tools are natural ways to assist this reasoning. A Tool–Raw comparison answers an engineering question—whether the complete assistance package improves accuracy—but not the scientific question of what caused the gain. The added prompt simultaneously changes semantic content, length, segmentation, location, authority, and sometimes the amount of computation left to the model. This distinction matters because models can preserve much of a prompt benefit under irrelevant or misleading instructions, randomized demonstration labels, and invalid chains of thought (Webson & Pavlick, 2022; Min et al., 2022; Wang et al., 2023); they are also sensitive to irrelevant context and formatting (Shi et al., 2023; Sclar et al., 2024; Yang et al., 2025). A higher Tool score therefore establishes utility, but not semantic attribution.

We develop source-verified matched controls that hold the program and prompt surface fixed while changing whether the added information is causally relevant to the answer. Each sham reports state that is true but disconnected from the queried computation. The core controls match annotation location, schema, words, characters, and lines. Two full-bank replications tighten the comparison: one matches complete prompts under each provider's native token counter, and one gives both arms

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.

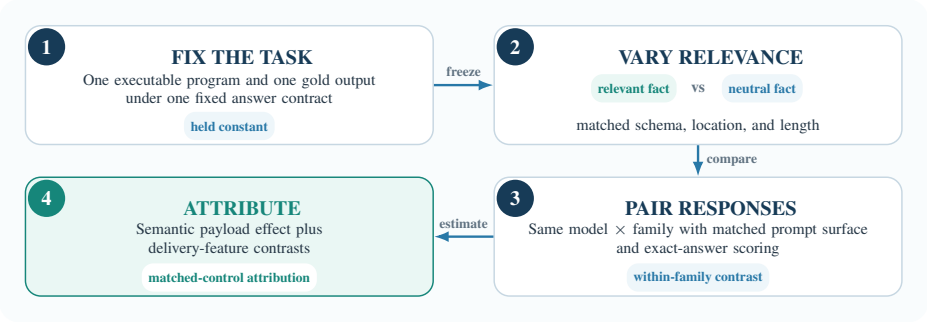


Figure 1: Matched-control attribution varies whether an added fact helps determine the answer while fixing the task and matching prompt-surface features. Further controls test how the identified payload is delivered.

the same neutral labels and field grammar. The resulting contrast asks whether useful semantic content beats an equally prominent, equally long, and equally truthful irrelevant note.

We apply the design to a Python information ladder and two deterministic, source-only tools. SemCheck recognizes nine Python semantic constructs and inserts applicable general rules. TraceCheck reconstructs visible-prefix Bash state while leaving a fixed suffix for the model. Same-bank controls then replace localized rules with an identical global rule header, selected rules with all nine rules, and the full trace with sufficient final-boundary state. Together, the two layers of controls first attribute the gain to useful semantic content and then test which delivery components add value.

Across 398 families and 6,608 calls to GPT, Claude, and Gemini model configurations, SemCheck and TraceCheck beat matched shams by +12.2 and +15.9 points. Exact-token replications retain +12.8 points in both languages, and the blind-label replications retain +10.6 and +12.0 points. Raw and sham baselines differ by less than three points, so Tool–Raw happens to estimate a similar magnitude on these banks. The matched design adds the missing attribution: the gain survives controls for length, layout, and explicit relevance cues.

A separate stress test composes 44 Python semantic stages and asks for 65 outputs. On Claude Opus 4.8, GPT-5.6 Sol, and Gemini 3.8 Flash, an exact snapshot after stage 39 raises pooled exact accuracy from 38.9% to 74.4%. This panel measures the value of resolved-state offloading under substantial remaining composition; a shared GPT-5.6 Terra configuration adds a +23.3-point common-model bridge.

The delivery controls sharpen the result. Localized rules do not outperform the same rules in a global header; selected rules do not outperform the full nine-rule panel; and the full TraceCheck presentation does not outperform sufficient plain state. Meanwhile, the corresponding global rules and plain state remain substantially better than matched shams. Thus the measurable benefit lies in the semantic payload rather than in the tested packaging. A separate information ladder adds an important boundary: general rules alone are not uniformly sufficient, whereas item-specific resolved values produce a large gain. Together, the studies favor simpler semantic assistance and identify when additional computation, rather than additional framing, is the valuable resource.

The tool-reliance evidence comprises two independently frozen analysis families on the same 40 model–family blocks. A three-condition causal-error study compares Correct, Inert-Error, and Causal-Error; a capability-and- verification extension evaluates Raw and Verify-Warning against the same Causal-Error arm. Opus 4.8 and GPT-5.6 Sol solve 92.5% of the programs unaided and remain perfect under correct or causally inert reports, yet every causal error redirects the response to its precomputed wrong output. Telling the model that the report may be wrong restores 97.5% accuracy at the same input-token count. The models possess the needed execution capability but do not invoke it by default against an authoritative intermediate state.

Our contributions are:

Raw
```
[no annotation]
box.__len__ = local
count = len(box)
direct = box.__len__()
```

Neutral-Sham (source-local)
```
# stage=1; role=note;
type=semantic_note;
topic=program_semantics;
detail={Binding data_4 contains
ordered literals (3,5,8)};
audit_pad=[matched]
box.__len__ = local
count = len(box)
direct = box.__len__()
```

Tool (source-local)
```
# stage=1; role=note;
type=semantic_note;
topic=program_semantics;
detail={Implicit len uses type slot;
explicit call uses instance value};
audit_pad=[matched]
box.__len__ = local
count = len(box)
direct = box.__len__()
```

Global-Rule (prompt header)
```
# stage=1; role=target;
kind=special_method_lookup;
name1=semantic_rule;
value1={Implicit len uses type slot;
explicit call uses instance value};
audit_pad=[matched]
...  [no local annotation] ...
box.__len__ = local
count = len(box)
```

Figure 2: Abridged presentations for one Python family. Tool and Neutral-Sham share the six displayed fields; only `detail` changes from disconnected state to the applicable rule. Global-Rule moves that rule to the header.

- causal evidence that applicable rule content on the SemCheck bank and resolved state in Bash improve execution prediction beyond prompt surface, replicated under exact provider-token and blind-label controls and complemented by a resolved-state stress test;
- a practical mechanism result: none of the tested delivery features— localization, rule filtering, or full-trace presentation—produces a supported average gain over a simpler presentation of the same useful information; and
- a reusable matched-control design and auditable evaluation package for code-semantic assistance, including provider-token and blind-label controls that attribute gains beyond the complete Tool–Raw bundle, plus two causal tool-reliance analyses that measure exact error propagation and prompted verification.

2 Matched-Control Attribution of Semantic Assistance

Task. We study exact execution prediction for programs whose outputs depend on language semantics that are present in the source but easy to misapply. A *family* contains one executable program or transcript and multiple presentations of the same computation. Without execution, the model returns a structured Python tuple or ordered Bash query outputs under a fixed exact-answer contract.

Why use a matched sham. Tool–Raw mixes semantic information with added tokens, segmentation, position, and salience. We instead compare useful annotations with *sham* annotations that are true of the source but causally irrelevant to the queried output. Within each family, informative and sham messages share locations, schema, line count, word count, and character count while executable code and gold output remain fixed. Tool minus Tool-Sham therefore identifies useful content; Raw measures the complete deployable package. Exact provider-token and blind-label replications additionally match native token counts and give both arms `role=note` with the same field grammar.

Exact provider-token control. Every SemCheck and TraceCheck family receives a full-bank token-matched replication. Tool is paired with a same-schema Token-Sham reporting a disconnected Python literal or unread Bash file. A final `audit_pad` equalizes complete-prompt counts for each model–family pair using `o200k_base` or the providers' official token counters.

Blind-label Neutral-Sham replication. Across all 60 SemCheck and 128 TraceCheck families, Tool and Neutral-Sham use identical neutral labels, schema, layout, and provider-native token counts. Both report source-verified state, but only Tool's state reaches the answer; Appendix B gives the construction.

2.1 STRONG-MODEL RESOLVED-STATE STRESS TEST

The stress test contains 30 disjoint Python families with 44 semantic stages and 65 requested ledger values. Five stages before completion, Tool reports the exact state after stage 39; Neutral-Sham reports a truthful local snapshot disconnected from the queried ledger. Notes match in schema, position, atom lengths, and provider-native tokens, and both arms require the same 65-value output. The paired estimand therefore measures resolved-state offloading, including its effect on completion. Opus 4.8 high, GPT-5.6 Sol medium, and Gemini 3.8 Flash high evaluate the 180-call grid; a 60-call GPT-5.6 Terra bridge links these families to the core panel.

2.2 CAUSAL TOOL ERROR AND VERIFICATION

The tool-reliance evidence uses 40 disjoint model–families: 20 depth-28 programs for Opus 4.8 high and 20 depth-44 programs for GPT-5.6 Sol medium, organized into two independently frozen analysis families. The three-condition causal-error study (120 calls) compares CORRECT, INERT-ERROR, and CAUSAL-ERROR. The capability-and-verification extension (80 calls) evaluates RAW and VERIFY-WARNING against the same Causal-Error arm. Depth is a fixed model-specific blocking variable that places unaided performance in an informative regime for each configuration. The pooled estimand averages within-model, within-family treatment effects with equal block weights across the two capability regimes. Five stages before completion, Raw omits the checkpoint, Correct reports exact state, Inert-Error corrupts one leaf that cannot reach the requested output, and CAUSAL-ERROR corrupts one leaf that changes two to four outputs. VERIFY-WARNING uses the same Causal-Error report but asks the model to check it against the program. Counterfactual execution supplies every induced wrong answer.

The three causal-error reports match tool identity, schema, position, length, and provider-native tokens; Verify-Warning also matches Causal-Error tokens by replacing audit padding. One Holm family compares Correct and Inert-Error with Causal-Error; a separately frozen Holm family compares Raw and Verify-Warning with Causal-Error. Equal-weight paired blocks use exact sign tests and a model-stratified bootstrap.

2.3 PYTHON INFORMATION LADDER

The Python ladder compares HIDDEN placeholders, a true but disconnected SHAM, an applicable general RULE, and the item-specific intermediate VALUE induced by that rule. Sham, Rule, and Value are surface matched; Value is diagnostic here, while the stress test makes resolved state its primary intervention.

2.4 AUTOMATIC SOURCE-DERIVED SCAFFOLDS

Python SemCheck. SemCheck detects nine declared Python semantic constructs with the abstract syntax tree and inserts each applicable fixed general rule at its source location; it neither executes the program nor computes item-specific values. The bank contains 36 compact three–five-mechanism families and 24 compositional 11–13-stage families covering all nine constructs.

Bash TraceCheck. TraceCheck applies hand-written transitions for nine Bash process and scope boundaries to reconstruct visible-prefix state. It inserts that state while leaving a fixed suffix of one, two, four, or five downstream stages for the model.

Both are deterministic source-only analyzers without gold answers or model responses. Their different outputs—localized Python rules and resolved Bash state—test the broader semantic-assistance principle.

2.5 SAME-BANK MECHANISM CONTROLS

Two same-bank controls vary delivery while preserving useful content. GLOBAL-RULE moves identical SemCheck rules from source-local comments to one header. PLAIN-STATE gives the sufficient final Bash boundary state as ordinary text while omitting earlier trace history and tool attribution. Executable inputs, gold outputs, and remaining computation stay fixed.

An `audit_pad` makes each control exactly token matched to Tool. Localized Tool minus Global-Rule and TraceCheck minus Plain-State form one Holm family. Global-Rule isolates localization conditional on selected rules.

Concurrent rule-selection control. In a nine-slot global header, SELECTED-RULES fills applicable slots and ALL-RULES fills all nine; prompts are token matched. The Selected-minus-All estimand uses the 36 families where the conditions differ.

2.6 ORACLE, MATCHING, AND INFERENCE

Python and Bash gold answers require agreement between two executions (CPython 3.14.3 or Bash 3.2.57) and an independent state-transition simulator. Builds reject nondeterminism, executable differences, unsafe commands, leakage, or failed matching; no human or language model labels an answer.

All scheduled calls enter intention-to-treat exact-answer scoring; unavailable, truncated, or unscorable responses are unsuccessful and retain their completion status. Primary intervals use 10,000 family-cluster bootstrap resamples, stratified by source and structural allocation for Python and by source and difficulty for Bash. Two-sided exact sign-flip tests swap paired treatments within family. Holm correction applies within each multi-hypothesis family; Appendix F gives the full inventory. A paper-wide Holm sensitivity over 17 primary hypotheses preserves 11 of 12 within-family supported effects; only the Terra bridge changes status (within-family $p = 0.0156$, paper-wide $p = 0.0938$).

3 EXPERIMENTAL SETUP

Frozen study banks. The formal package contains 6,608 answer calls over 398 unique families. Replications and mechanism controls reuse core families; the source-anchored, stress-test, and tool-reliance banks contribute 150 new families. Table 1 gives the inventory.

Analysis families. Core, exact-token, delivery, selection, source-anchored, blind-label, stress, bridge, fault-localization, and verification analyses each fix their items, estimands, and decision rules before their answer calls. Holm correction applies within each multi-hypothesis family.

Provider-balanced authorship. Core banks balance a reference generator with mechanism orders proposed by OpenAI, Anthropic, and Google models. Deterministic compilers instantiate all code and conditions; proposers never see literals, labels, or results.

Evaluated models. The core panel uses GPT-5.6 Terra medium, Claude Haiku 4.5, and Gemini 3.5 Flash high; the stress panel uses Claude Opus 4.8 high, GPT-5.6 Sol medium, and Gemini 3.8 Flash high. Terra bridges the two panels, and Opus plus Sol evaluate 20 tool-reliance families each. Every model receives the same exact-answer contract; order is shuffled by model, with sibling conditions nonadjacent. Appendix F records configurations and resolved identifiers.

Integrity and completion. All 6,608 scheduled responses enter scoring: 6,546 complete normally; 57 truncations, two API failures, one transport timeout, one empty, and one unparsable response are unsuccessful. The artifact verifies grids, hashes, and generated analyses; Appendix D gives details.

4 RESULTS

4.1 RULES ARE USEFUL BUT NOT UNIFORMLY SUFFICIENT

The Python information ladder scores 53.9% for Hidden, 54.4% for Sham, 58.3% for Rule, and 73.9% for Value. Rule adds +3.9 points over Sham (95% CI [-2.2, 10.0]), while item-specific Value adds +19.4 points. Resolved values produce the larger descriptive gain on this ladder and can help when a general rule alone leaves the consequence uncertain.

4.2 SemCheck rule content improves Python prediction

On the disjoint SemCheck bank, Raw accuracy is 51.7%, Tool-Sham is 50.0%, and SemCheck reaches 62.2%. The matched effect is +12.2 points (95% CI [+6.1, +18.9]; Holm-adjusted $p = 0.00264$), with 29 rescues and seven harms. Conventional SemCheck-minus-Raw is +10.6 points, close to the matched estimate because Raw and Tool-Sham differ by only -1.7 points. The ordinary comparison measures the gain accurately on this bank; the sham comparison shows that relevant rule content, rather than generic annotation, produces it.

The effect is positive for all three models, all four item-author sources, and both Python structural strata. Claude remains near floor (0.0% to 6.7%), while GPT and Gemini show larger gains.

The ladder and SemCheck results together reject both extremes. General rules are neither inert nor uniformly sufficient: they deliver a clear gain on the SemCheck bank, while the ladder shows that their value depends on the task and that resolved values can be substantially more effective.

4.3 Resolved-prefix state improves Bash prediction

The Bash study provides a cross-language content effect. Raw scores 68.8%, Tool-Sham 66.1%, and TraceCheck 82.0%. TraceCheck improves on its matched sham by +15.9 points (95% CI [+12.8, +19.0]; $p = 3.75 \times 10^{-16}$), with 63 rescues and two harms.

Claude gains +35.2 points and GPT +13.3, while Gemini is already at ceiling (100.0% to 99.2%). The effect is positive in every item-author source and difficulty band, with band-specific gains from +11.5 to +21.9 points.

4.4 The content effects survive exact token matching

Provider-token-matched replication preserves both effects. On Python, Tool scores 63.3% versus 50.6% for Token-Sham, a +12.8-point gain (95% CI [+7.2, +18.3]; Holm-adjusted $p = 0.00018$). On Bash, Tool scores 78.6% versus 65.9%, a +12.8-point gain (95% CI [+9.4, +16.4]; Holm-adjusted $p = 5.1 \times 10^{-10}$). Equalizing complete-prompt token counts therefore leaves the direction and magnitude intact.

4.5 The content effects survive blind labels

The Neutral-Sham replication removes `target`/`decoy` labels and phrases such as "no effect." Both arms use `role=note`, identical field grammar, newly collected responses, and symmetrically sampled audit codes. In Python, Tool scores 62.2% versus 51.7%, a +10.6-point gain (95% CI [+5.0, +16.7]; Holm-adjusted $p = 0.0022$). In Bash, Tool scores 79.2% versus 67.2%, a +12.0-point gain (95% CI [+8.9, +15.4]; Holm-adjusted $p = 2.7 \times 10^{-9}$).

Pooled effects remain positive across construct slices. Figure 3 summarizes the central pattern: content effects replicate across increasingly strict surface controls, whereas delivery-feature contrasts remain near zero.

The 80-family source-anchored replay is ceiling-limited: all arms reach 97.5%, with Tool-minus-Token-Sham at 0.0 points (95% CI [-2.5, +2.5]).

4.6 Models overtrust causal tool errors but verify when prompted

Correct checkpoints and one-leaf errors outside the answer path both achieve 100.0% exact accuracy across 40 model–family blocks. Moving the same kind of one-leaf error onto the causal path reduces accuracy to 0.0%: every response instead matches the counterfactual wrong answer produced by executing the remaining stages from that corrupted state. Across all 40 evaluated model–family blocks, both Correct-minus-Causal-Error and Inert-Error-minus-Causal-Error are +100.0 points, with 40 rescues and no harms. Their evaluated-bank family-bootstrap interval is [100.0, 100.0], with Holm-adjusted $p = 3.64 \times 10^{-12}$. Opus and Sol each reproduce the complete 20-family pattern at their fixed depths. The pooled estimand averages within-model, within-family treatment effects with equal block weights. With error sparsity, report format, position, length, and token count fixed, causal relevance determines whether a tool fault is harmless or fully controls the answer.

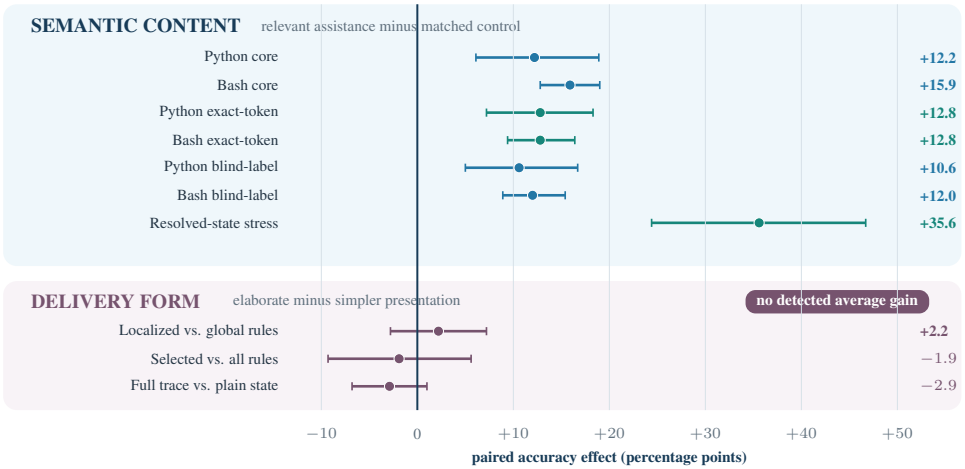


Figure 3: Paired exact-accuracy effects with 95% family-cluster bootstrap intervals. Content effects replicate; elaborate delivery shows no detected average gain over simpler presentations.

The same models solve 92.5% unaided under intention-to-treat scoring (37/40; 37/39 returned answers), a +92.5-point advantage over Causal-Error (95% CI [82.5, 100.0]; Holm-adjusted $p = 1.46\times10^{-11}$), establishing independent execution capability on 37 of the 40 evaluated blocks. An explicit instruction to verify the report restores 97.5% accuracy, a +97.5-point advantage over Causal-Error (95% CI [92.5, 100.0]; Holm-adjusted $p = 7.28\times10^{-12}$), without increasing provider-token count. This mitigation applies when the model has the independent capability demonstrated by Raw.

4.7 Resolved-state offloading improves the non-ceiling stress test

The 44-stage stress test requires the same 65-value output in both conditions. Under intention-to-treat scoring, the stage-39 snapshot raises exact accuracy from 38.9% to 74.4%, a +35.6-point effect (95% CI [+24.4, +46.7]; $p = 3.6\times10^{-6}$), with 35 rescues and 3 harms. Opus, Gemini, and Sol gain +43.3, +50.0, and +13.3 points, respectively. The treatment also reduces truncations from 35 to 17. The +35.6-point intention-to-treat estimate captures the total benefit of resolved-state offloading on correctness and completion under the fixed generation budget.

The Terra bridge holds a core-panel model fixed: accuracy falls from 68.3%/81.7% (Neutral-Sham/Tool) on core Python to 0.0%/23.3% on the stress bank. Its +23.3-point paired effect (95% CI [+10.0, +40.0]) provides a common-model link between panels: the stress bank is harder for Terra, yet resolved-state assistance remains beneficial.

4.8 Useful content, not elaborate packaging, explains the gains

Appendix Table 5 compares each tool with a simpler presentation of its useful content. On the same 60 Python families, localized Tool reaches 63.3% and the identical global rule header reaches 61.1%, a +2.2-point localized-minus-global difference (95% CI [-2.8, +7.2]; Holm-adjusted $p = 0.587$). Global-Rule remains +10.6 points above Token-Sham: the SemCheck gain survives without source-local placement. The concurrent selection control reaches 56.5% with selected rules and 58.3% with all nine: -1.9 points (95% CI [-9.3, +5.6]; $p = 0.815$). On this panel, the full nine-rule header retains performance without rule selection.

The Bash result is sharper. Full TraceCheck reaches 78.6%, while sufficient Plain-State reaches 81.5%; the full-trace-minus-plain-state contrast is -2.9 points (95% CI [-6.8, +1.0]; Holm-adjusted $p = 0.402$). Plain-State itself is +15.6 points above Token-Sham. Thus deterministic partial-execution offloading explains the Bash gain, while trace history and tool attribution add no measured average benefit. TraceCheck remains operationally useful as the source-only procedure that computes the sufficient state.

Across all three controls, simpler delivery retains the tested gains. The intervals still allow a +7.2-point localization effect and a +5.6-point selection effect on other samples, but bound any positive full-trace effect on this panel to +1.0 point. The actionable finding is therefore to prioritize correct semantic payload before adding localization, filtering, or trace presentation.

5 Related Work

Form, content, and placebo controls. Prompt gains can survive irrelevant instructions (Webson & Pavlick, 2022), randomized demonstration labels (Min et al., 2022), and invalid chain-of-thought examples (Wang et al., 2023); irrelevant context and formatting also alter accuracy (Shi et al., 2023; Yang et al., 2025; Sclar et al., 2024; Voronov et al., 2024). Matched controls have separated explanatory content from lower-level prompt features in in-context learning (Lampinen et al., 2022); recent concept-scaffold work compares helpful, format-only, misleading, and wrong-fact controls in mathematics (He et al., 2026). Placebo controls have further reached frozen code-model self-repair (Iscan, 2026). We extend this line to exact code execution with source-verified truthful counterfactuals, provider-token equality, blind labels, and same-bank interventions over localization, selection, and trace presentation.

Code execution and semantic assistance. CRUXEval measures executable Python prediction (Gu et al., 2024); CodeSense, TraceEval, and PLSemanticsBench broaden semantic evaluation across projects, languages, and constructs (Roy et al., 2026; Li et al., 2026; Thimmaiah et al., 2026). Execution-trace methods (Ni et al., 2024; Armengol-Estapé et al., 2025; Wang et al., 2025) and code-induced reasoning (Waheed et al., 2026) study whether structure or traces help. We isolate which component helps by holding the executable task fixed and using synthetic counterfactuals with exact dual oracles.

Faulty tools and verification. Prior work documents erroneous-tool acceptance and studies detection and recovery from silent failures, including prompts that ask models to verify tool outputs (Ye et al., 2024; Sun et al., 2024). Our executable intervention holds report surface fixed, separates causally inert from output-changing errors, and tests token-matched prompted verification against a precomputed counterfactual wrong answer.

6 Limitations

Controlled attribution versus natural transfer. Synthetic families are a design asset for this causal question: they permit executable counterfactuals, exact dual oracles, and identical programs across conditions. The estimates cover the tested mechanisms and prompt contracts; the 97.5% source-anchored replay motivates harder dependency-preserving repository transfer tests.

Prompt and model scope. The full-bank blind-label study uses symmetric 3,072-character audit codes for exact token equality; the resolved-state stress test provides a naturally matched 64-character alternative. GPT-5.6 Terra spans the core and stress banks; the other five configurations are panel-specific and not compute-matched. The executable scope is CPython 3.14.3 and macOS Bash 3.2.57, with nine constructs per tool. Broader languages, runtimes, and repository-scale tasks define the next transfer tests.

7 Conclusion

Matched controls attribute SemCheck gains to applicable rules and TraceCheck gains to resolved state; exact-token and blind-label replications preserve both. Matched one-leaf tool faults further show direct semantic reliance in Claude Opus 4.8 and GPT-5.6 Sol: causal errors redirect every response to the corresponding wrong output. In the separately frozen capability-and- verification analysis on the same blocks, the two configurations solve 92.5% unaided, and an explicit verification instruction restores 97.5% accuracy at the same input-token count. The 44-stage stress test yields a +35.6-point effect across Opus 4.8, Gemini 3.8 Flash, and GPT-5.6 Sol; a common-model bridge connects the core and stress panels. Global rules retain the SemCheck gain, while rule selection

adds no supported average gain over all nine rules and plain state captures the Bash gain without a full trace. Semantic assistance should prioritize correct payload and resolved computation before elaborate packaging.

AI use statement. LLMs were subjects, constrained item proposers, and assistants. OpenAI, Anthropic, and Google models proposed mechanism orders used to construct part of the synthetic benchmark; deterministic compilers instantiated all items, executable oracles supplied every gold answer, and no LLM graded responses. Assistants helped with brainstorming, literature search, coding, and manuscript drafting. The authors made and verified all final research, design, analysis, interpretation, and conclusion decisions and take full responsibility.

Ethics statement. No human participants or personal data were used; execution was safety-validated and outputs follow provider terms.

Reproducibility statement. The anonymous artifact provides evaluated prompts, gold answers, raw responses, tool and analysis code, source provenance, and integrity hashes at `https://anonymous.4open.science/r/matched-control-code-reasoning-artifact-75AB`.

REFERENCES

Jordi Armengol-Estapé, Quentin Carbonneaux, Tianjun Zhang, Aram H. Markosyan, Volker Seeker, Chris Cummins, Melanie Kambadur, Michael F. P. O'Boyle, Sida Wang, Gabriel Synnaeve, and Hugh James Leather. What I cannot execute, I do not understand: Training and evaluating LLMs on program execution traces. *arXiv preprint arXiv:2503.05703*, 2025. URL https://arxiv.org/abs/2503.05703.

Alex Gu, Baptiste Roziere, Hugh James Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida Wang. CRUXEval: A benchmark for code reasoning, understanding and execution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 16568–16621, 2024. URL https://proceedings.mlr.press/v235/gu24c.html.

Jiangshan He, Song Dai, Xiaolong Qiao, Jungang Li, Yibo Yan, and Xuming Hu. Beyond direct gains: Matched controls for evaluating concept scaffolds. In *3rd AI for Math Workshop at ICML*, 2026. URL https://openreview.net/forum?id=R3ltJC0uZN.

Mehmet Iscan. Form, not content? A preregistered, placebo-controlled evaluation of learned error-conditioned self-repair through prompts and weights in frozen small code models. *arXiv preprint arXiv:2607.12962*, 2026. URL https://arxiv.org/abs/2607.12962.

Andrew Lampinen, Ishita Dasgupta, Stephanie Chan, Kory Mathewson, Mh Tessler, Antonia Creswell, James McClelland, Jane Wang, and Felix Hill. Can language models learn from explanations in context? In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 537–563. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.findings-emnlp.38. URL https://aclanthology.org/2022.findings-emnlp.38/.

Yikun Li, Jinfeng Jiang, Ting Zhang, Chengran Yang, Chenxing Zhong, Yide Yin, Wen Bin Leow, Eng Lieh Ouh, Lwin Khin Shar, and David Lo. An execution-verified multi-language benchmark for code semantic reasoning. *arXiv preprint arXiv:2605.11006*, 2026. URL https://arxiv.org/abs/2605.11006.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11048–11064. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.emnlp-main.759. URL https://aclanthology.org/2022.emnlp-main.759/.

Ansong Ni, Miltiadis Allamanis, Arman Cohan, Yinlin Deng, Kensen Shi, Charles Sutton, and Pengcheng Yin. NExT: Teaching large language models to reason about code execution. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2404.14662.

Monoshi Kumar Roy, Simin Chen, Benjamin Steenhoek, Jinjun Peng, Gail Kaiser, Baishakhi Ray, and Wei Le. CodeSense: A real-world benchmark and dataset for code semantic reasoning. *Inter-*

national Conference on Learning Representations, 2026. URL https://arxiv.org/abs/2506.00750.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/6c0e99d736da621403018ca7b32b1a4d-Abstract-Conference.html.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31210–31227, 2023. URL https://proceedings.mlr.press/v202/shi23a.html.

Jimin Sun, So Yeon Min, Yingshan Chang, and Yonatan Bisk. Tools fail: Detecting silent errors in faulty tools. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 14272–14289. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.790. URL https://aclanthology.org/2024.emnlp-main.790/.

Aditya Thimmaiah, Jiyang Zhang, Jayanth Srinivasa, Junyi Jessy Li, and Milos Gligoric. PLSemanticsBench: Large language models as programming language interpreters. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2510.03415.

Anton Voronov, Lena Wolf, and Max Ryabinin. Mind your format: Towards consistent evaluation of in-context learning improvements. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 6287–6310. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.375. URL https://aclanthology.org/2024.findings-acl.375/.

Abdul Waheed, Zhen Wu, Carolyn Rose, and Daphne Ippolito. On code-induced reasoning in LLMs. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/f39cbd56877419229aef90dbe463c182-Abstract-Conference.html.

Boshi Wang, Sewon Min, Xiang Deng, Jiaming Shen, You Wu, Luke Zettlemoyer, and Huan Sun. Towards understanding chain-of-thought prompting: An empirical study of what matters. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2717–2739. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.153. URL https://aclanthology.org/2023.acl-long.153/.

Jian Jornbowrl Wang, Xiaofei Xie, Qiang Hu, Shangqing Liu, and Yi Li. Do code semantics help? A comprehensive study on execution trace-based information for code large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 10367–10385, 2025. doi: 10.18653/v1/2025.findings-emnlp.548. URL https://aclanthology.org/2025.findings-emnlp.548/.

Albert Webson and Ellie Pavlick. Do prompt-based models really understand the meaning of their prompts? In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2300–2344. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.naacl-main.167. URL https://aclanthology.org/2022.naacl-main.167/.

Minglai Yang, Ethan Huang, Liang Zhang, Mihai Surdeanu, William Yang Wang, and Liangming Pan. How is LLM reasoning distracted by irrelevant context? An analysis using a controlled benchmark. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13329–13347. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.674. URL https://aclanthology.org/2025.emnlp-main.674/.

Junjie Ye, Sixian Li, Guanyu Li, Caishuang Huang, Songyang Gao, Yilong Wu, Qi Zhang, Tao Gui, and Xuanjing Huang. ToolSword: Unveiling safety issues of large language models in tool learning across three stages. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2181–2211. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.119. URL https://aclanthology.org/2024.acl-long.119/.

A EXPANDED RESULTS

Table 1: Formal study inventory. Replications and mechanism controls reuse core families; the total counts unique executable families once. For the two tool-reliance rows, 1^* means one model per family across two model configurations (20 families each); the extension reuses the same 40 blocks.

Study	Families	Conditions	Models	Calls	Role
Python information ladder	60	4	3	720	core, disjoint
Python SemCheck	60	3	3	540	core tool
Bash TraceCheck	128	3	3	1,152	core tool
Exact-token replication	188	2	3	1,128	reused families
Mechanism controls	188	–	3	780	eligible reused families
Source-anchored ceiling probe	80	3	3	720	new families
Blind-label replication	188	2	3	1,128	reused; new responses
Resolved-state stress test	30	2	3	180	new Python families
Terra cross-panel bridge	30	2	1	60	reused stress families
Causal-error study	40	3	1^*	120	new model-specific families
Capability/verification extension	40	2	1^*	80	reused causal blocks
Total	**398**	–	–	**6,608**	unique families counted once

Table 2 places the automatic-tool effects, their exact provider-token and blind-label replications, and the source-anchored boundary result on a common percentage-point scale. The resolved-state stress test uses disjoint families and appears in Table 3; the Terra bridge reuses those stress families as a common-model comparison. Replications reuse the corresponding core families and therefore add responses rather than new executable families.

Table 2: Core automatic-tool, robustness, and boundary effects. The intervals use family-cluster bootstrap resampling.

Study	Contrast	Δ (pp)	95% CI low	95% CI high	Supported
Python core	SemCheck - Tool-Sham	12.2	6.1	18.9	Yes
Python token robustness	Tool - Token-Sham	12.8	7.2	18.3	Yes
Bash core	TraceCheck - Tool-Sham	15.9	12.8	19.0	Yes
Bash token robustness	Tool - Token-Sham	12.8	9.4	16.4	Yes
Python blind-label robustness	Tool - Neutral-Sham	10.6	5.0	16.7	Yes
Bash blind-label robustness	Tool - Neutral-Sham	12.0	8.9	15.4	Yes
Source-anchored replay	Tool - Token-Sham	0.0	-2.5	2.5	No
Terra cross-panel bridge	Tool - Neutral-Sham	23.3	10.0	40.0	Yes

A.1 RESOLVED-STATE STRESS TEST BY MODEL

All three strong-model configurations improve with the stage-39 snapshot (Table 3). Opus and Gemini have the largest headroom; Sol begins at 80.0% and still gains 13.3 points. The pooled estimate uses equal model–family weight and family-clustered inference. Omitting Opus, Gemini, or Sol yields pooled gaps of +31.7, +28.3, and +46.7 points. The separate Terra bridge links the panels directly: the same configuration scores 68.3%/81.7% (Neutral-Sham/Tool) on the core Python blind-label bank and 0.0%/23.3% on the stress bank.

Table 3: Resolved-state Python stress test. Accuracies and differences are percentages; rescues and harms are paired exact-answer transitions. Terra is a separate cross-panel bridge and is not included in the pooled row.

Model	Neutral-Sham	Tool	Δ (pp)	Rescues	Harms
Claude Opus 4.8 high	0.0	43.3	+43.3	13	0
Gemini 3.8 Flash high	36.7	86.7	+50.0	16	1
GPT-5.6 Sol medium	80.0	93.3	+13.3	6	2
Pooled	**38.9**	**74.4**	**+35.6**	**35**	**3**
GPT-5.6 Terra medium (bridge)	0.0	23.3	+23.3	7	0

Completion is part of the exact-answer outcome. Tool has 73 normal completions and 17 truncations; Neutral-Sham has 53 normal completions, 35 truncations, one empty response, and one unparsable response. Opus contributes 14 versus six normal completions (Tool versus Neutral-Sham), Gemini 29 versus 17, and Sol 30 versus 30, making Opus's 0.0% Neutral-Sham accuracy completion-limited. Grading every available answer yields a +33.3-point gap. Among the 49 pairs with two normal completions, the descriptive gap is +22.4 points (95% CI [6.1, 38.8]). The causal estimand is the intention-to-treat gap, which includes the treatment's effect on completion.

A.2 Causal tool reliance by model

Table 4 reports the shared blocks from both independently frozen tool-reliance analyses. The three-condition causal-error study has 120 normal, parsed responses; the capability-and-verification extension has 79 normal responses and one Raw transport timeout. In the Causal-Error arm, each response matches the counterfactual answer obtained by executing the five-stage suffix from the corrupted checkpoint.

Table 4: Causal tool-reliance analyses under intention-to-treat scoring. Opus uses depth 28 and Sol depth 44; all contrasts are within model–family blocks. Entries are exact accuracy percentages; Wrong sig. is the percentage matching the precomputed output induced by the causal checkpoint error. Verify uses that same erroneous report with an explicit instruction to check it.

Model	Raw	Correct	Inert	Causal	Wrong sig.	Verify
Claude Opus 4.8 high	95.0	100.0	100.0	0.0	100.0	95.0
GPT-5.6 Sol medium	90.0	100.0	100.0	0.0	100.0	100.0
Pooled	**92.5**	**100.0**	**100.0**	**0.0**	**100.0**	**97.5**

Table 5: Same-bank delivery controls. Values and family-cluster intervals are percentages; n counts paired model–families, and rescues/harms are paired transitions.

Setting	Mechanism contrast	n	Left	Right	Δ (pp)	Resc.	Harms	95% CI
Python	Localized Tool − Global rule	180	63.3	61.1	+2.2	15	11	[-2.8, +7.2]
Python	Selected rules − All nine	108	56.5	58.3	-1.9	10	12	[-9.3, +5.6]
Bash	TraceCheck − Plain state	384	78.6	81.5	-2.9	22	33	[-6.8, +1.0]

A.3 Core effects by model

Table 6 shows the model heterogeneity behind the two core pooled effects. SemCheck improves GPT and Gemini while Claude remains near floor (0.0% to 6.7%). TraceCheck improves Claude and GPT while Gemini is at ceiling. Each language therefore contains two responsive models and a different boundary model.

Table 6: Core automatic-tool results by model. Accuracies and paired differences are percentages.

Study	Model	n	Tool-Sham (%)	Tool (%)	Δ (pp)	Rescues	Harms
Python SemCheck	Claude Haiku 4.5	60	0.0	6.7	+6.7	4	0
	Gemini 3.5 Flash	60	85.0	96.7	+11.7	8	1
	GPT-5.6 Terra	60	65.0	83.3	+18.3	17	6
Bash TraceCheck	Claude Haiku 4.5	128	13.3	48.4	+35.2	46	1
	Gemini 3.5 Flash	128	100.0	99.2	-0.8	0	1
	GPT-5.6 Terra	128	85.2	98.4	+13.3	17	0

A.4 Exact-token robustness by model

Table 7 resolves the exact-token effects by model. Python gains are positive for all three models. In Bash, Claude and GPT improve while Gemini remains at ceiling. Rescues are Token-Sham errors corrected by Tool; harms are the reverse.

Table 7: Provider-token-matched robustness by model. Accuracies and differences are percentages.

Language	Model	n	Token-Sham	Tool	Δ (pp)	Rescues	Harms
Python	Claude Haiku 4.5	60	0.0	3.3	+3.3	2	0
Python	Gemini 3.5 Flash	60	81.7	96.7	+15.0	11	2
Python	GPT-5.6 Terra	60	70.0	90.0	+20.0	15	3
Shell	Claude Haiku 4.5	128	10.9	39.8	+28.9	38	1
Shell	Gemini 3.5 Flash	128	100.0	100.0	+0.0	0	0
Shell	GPT-5.6 Terra	128	86.7	96.1	+9.4	15	3

Construct and omission sensitivity. Table 8 reports the exact-token Tool–Token-Sham gap within every construct-containing slice. Families can contain multiple constructs, so rows overlap and are descriptive rather than separate hypothesis tests. Every pooled construct slice is positive. The gap remains positive after omitting any one model (Python: +9.2 to +17.5 points; Bash: +4.7 to +19.1) or every family containing any one construct (Python: +9.8 to +19.6; Bash: +9.5 to +25.0). Model-by-construct cells retain the expected floor/ceiling heterogeneity.

Table 8: Descriptive exact-token heterogeneity. Entries are paired Tool-minus-Token-Sham percentage-point gaps among families containing the named construct; construct slices overlap.

Language	Construct	Families	Claude	Gemini	GPT	Pooled
Python	context suppression	38	+2.6	+15.8	+15.8	+11.4
	cooperative MRO	43	+2.3	+16.3	+23.3	+14.0
	decorator order	39	0.0	+15.4	+15.4	+10.3
	descriptor precedence	42	0.0	+14.3	+16.7	+10.3
	mapping protocols	38	+2.6	+13.2	+13.2	+9.6
	match-guard binding	36	+2.8	+16.7	+11.1	+10.2
	metaclass namespace	40	0.0	+22.5	+17.5	+13.3
	special-method lookup	43	0.0	+18.6	+11.6	+10.1
	`yield from` protocol	41	+4.9	+14.6	+22.0	+13.8
Bash	brace group	113	+27.4	0.0	+6.2	+11.2
	command substitution	109	+27.5	0.0	+7.3	+11.6
	launch helper	112	+25.9	0.0	+7.1	+11.0
	nested substitution	109	+27.5	0.0	+8.3	+11.9
	pipeline loop	110	+26.4	0.0	+9.1	+11.8
	pipeline source	105	+28.6	0.0	+6.7	+11.7
	read redirect	114	+29.8	0.0	+9.6	+13.2
	source helper	113	+26.5	0.0	+8.8	+11.8
	subshell file	107	+28.0	0.0	+7.5	+11.8

A.5 Blind-label robustness by model and construct

Table 9 reports paired outcomes from the blind-label study. Python is positive for all three models. In Bash, Claude has substantial headroom and improves strongly, GPT has a small positive gap, and Gemini is unchanged at 100%. The pooled estimand weights every model–family cell equally.

Table 9: Blind-label Neutral-Sham replication by model. Accuracies and paired differences are percentages.

Language	Model	n	Neutral-Sham	Tool	Δ (pp)	Rescues	Harms
Python	Claude Haiku 4.5	60	0.0	5.0	+5.0	3	0
Python	Gemini 3.5 Flash	60	86.7	100.0	+13.3	8	0
Python	GPT-5.6 Terra	60	68.3	81.7	+13.3	16	8
Bash	Claude Haiku 4.5	128	10.2	43.0	+32.8	45	3
Bash	Gemini 3.5 Flash	128	100.0	100.0	+0.0	0	0
Bash	GPT-5.6 Terra	128	91.4	94.5	+3.1	9	5

Table 10 gives overlapping construct-containing slices as a descriptive heterogeneity analysis. All 18 pooled construct gaps are positive. Omitting any one model leaves gaps of +9.2 to +13.3 points in Python and +1.6 to +18.0 in Bash; omitting every family containing any one construct leaves +4.2 to +13.6 and +4.8 to +20.0. Individual model-by-construct cells show floor and ceiling effects.

Table 10: Descriptive blind-label heterogeneity. Entries are paired Tool-minus-Neutral-Sham percentage-point gaps among families containing the named construct; construct slices overlap.

Language	Construct	Families	Claude	Gemini	GPT	Pooled
Python	context suppression	38	+2.6	+18.4	+7.9	+9.6
	cooperative MRO	43	+4.7	+14.0	+16.3	+11.6
	decorator order	39	+0.0	+15.4	+17.9	+11.1
	descriptor precedence	42	+0.0	+19.0	+16.7	+11.9
	mapping protocols	38	+0.0	+18.4	+7.9	+8.8
	match-guard binding	36	+8.3	+19.4	+16.7	+14.8
	metaclass namespace	40	+0.0	+20.0	+12.5	+10.8
	special-method lookup	43	+0.0	+16.3	+16.3	+10.9
	`yield from` protocol	41	+7.3	+17.1	+12.2	+12.2
Bash	brace group	113	+32.7	+0.0	+0.0	+10.9
	command substitution	109	+32.1	+0.0	+0.0	+10.7
	launch helper	112	+33.9	+0.0	+0.9	+11.6
	nested substitution	109	+33.9	+0.0	+0.0	+11.3
	pipeline loop	110	+34.5	+0.0	+0.9	+11.8
	pipeline source	105	+33.3	+0.0	-1.0	+10.8
	read redirect	114	+35.1	+0.0	+3.5	+12.9
	source helper	113	+34.5	+0.0	+0.9	+11.8
	subshell file	107	+31.8	+0.0	+0.0	+10.6

A.6 Mechanism controls by model

Table 11 shows the heterogeneity behind the two pooled mechanism tests. Python localization is positive only for GPT, while Claude and Gemini slightly favor the global presentation. In Bash, GPT slightly favors TraceCheck, Gemini is unchanged at ceiling, and Claude favors Plain-State. In the concurrent Python selection control, Claude favors selected rules by 2.8 points, Gemini is unchanged, and GPT favors all nine by 8.3 points.

Table 11: Same-bank mechanism controls by model. Left-minus-right contrasts are localized Tool minus Global-Rule, TraceCheck minus Plain-State, and Selected-Rules minus All-Rules. Values are exact accuracies and paired differences in percent.

Setting	Model	Left	Right	Δ
Python	Claude	3.3	5.0	-1.7
Python	Gemini	96.7	100.0	-3.3
Python	GPT	90.0	78.3	+11.7
Bash	Claude	39.8	50.8	-10.9
Bash	Gemini	100.0	100.0	+0.0
Bash	GPT	96.1	93.8	+2.3
Python selection	Claude	8.3	5.6	+2.8
Python selection	Gemini	97.2	97.2	+0.0
Python selection	GPT	63.9	72.2	-8.3

A.7 Source-anchored replay by model

Table 12 exposes the ceiling behind the pooled null. Language-specific Tool-minus-Token-Sham gaps are +0.8 points in Python and -0.8 in Bash. The corresponding upstream- and downstream-provenance gaps are -1.7 and +1.7 points.

Table 12: Source-anchored replay by model. Tool denotes ReplayCheck.

Model	n	Raw	Token-Sham	Tool	Δ (pp)	Rescues	Harms
Claude Haiku 4.5	80	92.5	93.8	95.0	+1.2	4	3
Gemini 3.5 Flash	80	100.0	98.8	98.8	+0.0	1	1
GPT-5.6 Terra	80	100.0	100.0	98.8	-1.2	0	1

B Condition Construction and Answer Contract

Every family fixes the executable program or displayed command transcript, task instructions, answer schema, and gold output before rendering conditions. Only a non-executable annotation slot changes. Raw leaves the slot empty; Sham reports true state with no causal path to the queried output; Rule or Tool reports task-relevant semantic information. The original controls match annotation locations, field structure, lines, lexical words, and characters. The robustness, mechanism, replay, and Neutral-Sham controls additionally match the complete prompt's provider-native token count separately for each model-family pair. Only the final `audit_pad` field may vary to obtain equality.

For the blind-label replication, both arms use `role=note`, identical field names, and the same line layout. Both audit codes are searched symmetrically from a deterministic SHA-256 distribution and contain exactly 3,072 lowercase hexadecimal characters in 48 fixed-width lines. The frozen serialized prompts have zero provider-token mismatches across model–family pairs. The 3,072-character audit pad defines this controlled robustness format.

The resolved-state stress test achieves equality without differential padding. Tool and Neutral-Sham snapshots match in schema, position, atom count, and atom-length multiset; paired prompts have equal provider-native token counts and share the same 64-character lowercase hexadecimal audit identifier.

The three checkpoint-manipulation conditions use the same report schema. Correct contains the exact checkpoint; Inert-Error and Causal-Error each replace one generated leaf through the same deterministic replacement function. Counterfactual execution verifies that the inert replacement cannot reach the requested output and that the causal replacement changes two to four requested values. Complete prompts match exactly in characters, words, source program, and provider-native tokens within every model–family block. Verify-Warning retains the Causal-Error report and provider-token count while replacing neutral audit padding with a verification instruction; Raw presents the identical source without a checkpoint.

Every Bash prompt explicitly specifies a single non-interactive `/bin/bash` 3.2.57 process on macOS. The Bash results target this runtime.

All prompts require one final line beginning with `ANSWER:`. A frozen parser extracts that field, removes permitted wrapper whitespace, and applies the task-specific structured-answer parser. No judge model is used. Refusal, empty output, an invalid schema, a response ending at its maximum output cap, or terminal API failure scores as unsuccessful under intention-to-treat exact-answer scoring; its completion state remains separately reported.

Information boundaries. SemCheck emits a source-localized general Python rule and never computes item state. TraceCheck emits state after the visible Bash prefix but leaves a fixed suffix unresolved. Global-Rule moves SemCheck's identical rule text to a single header. Selected-Rules fills only applicable slots in a fixed nine-slot header, whereas All-Rules fills every slot; neither localizes rules. Plain-State reports TraceCheck's sufficient final boundary as ordinary problem text without tool attribution. ReplayCheck emits one intermediate dispatch, binding, object, process, file, or environment checkpoint and likewise leaves a downstream suffix. All analyzers are deterministic source-only functions of the displayed task and receive neither gold outputs nor model responses.

For the formal SemCheck bank, the builder rejects any family whose detected construct multiset differs from the declared generator components. Expected- instance recognition is therefore 100% on retained synthetic items. Natural- repository recognition lies outside this evaluation.

C ITEM-SOURCE BALANCE

Core banks balance four item sources: a reference generator and mechanism orders proposed from byte-identical prompts to OpenAI, Anthropic, and Google models. Proposals are closed-IR sequences over nine declared semantic IDs, not code or answers. Python orders contain 3–5 distinct IDs or 11–13 stages covering all nine; Bash bands contain 4, 9, 16, or 20 stages under fixed coverage and repetition caps. One deterministic compiler instantiates every transition and condition, and salted-hash selection applies duplicate, overlap, and validity gates without performance data. Each Python bank has 15 families per source; Bash has eight per source-by-band cell.

D ORACLE, SAFETY, AND LOCK SEQUENCE

Python core items require byte-identical output from two fresh CPython 3.14.3 runs and agreement with an independently implemented state-transition model. Bash core items analogously require two Bash 3.2.57 runs and a typed state simulator. Source-anchored replays execute twice in fresh temporary directories and ReplayCheck must independently recover the frozen checkpoint from source alone. Perturbing or deleting the unresolved suffix must change or make the final answer unavailable. Builds reject nondeterminism, annotation leakage, executable differences between conditions, unsafe commands, and a gold answer copied into the prompt.

The formal sequence is:

1. freeze protocols, model configuration, power analysis, construction rules, and selection salts;
2. build surplus candidates and apply outcome-blind oracle, leakage, overlap, and safety gates;
3. select fixed quotas by salted hash, generate every presentation, and freeze items, schedules, tools, prompts, parsers, and analysis code;
4. count and match provider-native tokens where applicable, then freeze the corresponding prompt bank before the study's scheduled answer calls;
5. execute the complete shuffled grid without condition-wise result inspection, retain every scheduled failure, and run the frozen analysis; and
6. audit design and item hashes, response-grid completeness, resolved provider model identifiers, and generated result hashes.

Across the full artifact, 6,546 of 6,608 scheduled responses complete normally. Intention-to-treat scoring includes 57 truncations, two terminal API failures, one transport timeout, one empty response, and one unparsable response. The resolved-state stress test accounts for 52 truncations, the empty response, and the unparsable response; the other studies account for five truncations and two API failures. Every scheduled model–item identity appears exactly once in the analysis projection; all 60 Terra bridge and 120 checkpoint-manipulation responses complete normally.

E SOURCE-ANCHORED REPLAY PROVENANCE

The ceiling probe anchors 40 Python and 40 Bash families to immutable source occurrences from each language implementation's regression suite and four downstream projects. Each family has Raw, Token-Sham, and ReplayCheck presentations; ReplayCheck leaves a downstream suffix unresolved, and paired Tool/Sham prompts are provider-token matched.

The source-structure-derived replay compiler preserves each detected semantic operator and its local dispatch, scope, binding, delegation, or class- construction topology; replaces project dependencies with visible deterministic stand-ins; and derives literals from the salted source identity. This construction isolates language semantics from unavailable library knowledge. Item selection, token counts, and schedules are locked before answer-model calls. A deterministic pipeline constructs, filters, labels, and grades every retained replay. Table 13 gives the locked repository balance.

Table 13: Locked replay provenance. Every retained family has a distinct immutable source occurrence; full commit hashes, paths, line spans, and fragment hashes are included in the artifact.

Repository	Language	Provenance	Families	License
CPython	Python	upstream regression	20	PSF-2.0
attrs	Python	downstream project	5	MIT
pytest	Python	downstream project	5	MIT
Pydantic	Python	downstream project	5	MIT
Django	Python	downstream project	5	BSD-3-Clause
Bash	Bash	upstream regression	20	GPL-3.0-or-later
bats-core	Bash	downstream project	5	MIT
pyenv	Bash	downstream project	5	MIT
asdf	Bash	downstream project	5	MIT
Bash-it	Bash	downstream project	5	MIT

The 40 Python families cover seven replay groups: decorators, method resolution, pattern matching, delegation with `yield from`, metaclass construction, context suppression, and descriptors. The 40 Bash families cover six groups: subshells, brace groups, pipelines, command substitution, sourcing, and redirected reads. No group exceeds eight families in one language, upstream families span multiple files, and no source file contributes more than five families.

F Model Panel and Inference Details

The two-language panel consists of GPT-5.6 Terra with medium reasoning, dated Claude Haiku 4.5, and Gemini 3.5 Flash with high thinking and temperature zero. GPT uses low verbosity; temperature is unset for GPT and Claude, and top-p and frequency penalty are not supplied. Adaptive output caps are 16k/32k for GPT and Claude and 32k/65,536 for Gemini. The resolved-state stress test uses Claude Opus 4.8 high, GPT-5.6 Sol medium, and Gemini 3.8 Flash high. Provider-returned identifiers are retained and stable across collections; sibling conditions are separated in a model-specific frozen random order. The cross-panel bridge applies the core panel's GPT-5.6 Terra medium configuration to the same 30 stress families. The causal-error study and capability-and-verification extension use Opus 4.8 high at depth 28 and GPT-5.6 Sol medium at depth 44, with five stages remaining after the reported checkpoint.

Every model–family cell receives equal weight; the three paired model differences are averaged within family for inference. Exact two-sided sign-flip tests swap treatments jointly within family, and intervals use 10,000 pre-specified stratified family-bootstrap samples. The two Python core hypotheses are Holm-adjusted together, as are the two exact-token robustness hypotheses. The two delivery-control hypotheses form a third Holm family. The concurrent rule-selection control, Bash core, and source-anchored ceiling probe each have one separately frozen primary test. The two blind-label hypotheses form a separately frozen Holm family. The resolved-state stress test and Terra bridge each have one family-clustered primary test. The fault-localization and verification probes each form a two-contrast Holm family and use equal-weight model–family blocks with bootstrap strata by model. Model, source, repository, mechanism, and difficulty-band slices characterize heterogeneity.