IMPLEMENTATION-SPECIFIC EFFECTS IN LANGUAGE-MODEL HARNESSES

Anonymous authors
Paper under double-blind review

ABSTRACT

Runtime code determines which context a language model receives, how often it is called, and when its answers are revised. **HarnessBench** studies these choices through 2^4 factorial ablations of context management, memory, feedback, and planning. Each component has a shared null and two concrete active implementations. A complete-output replication contains 4,608 evaluations across three fixed model identifiers, two task domains, 16 configurations, and both implementation variants. Direct paired contrasts show differences in marginal effects between global implementation bundles, including Planning on math and code and Context and Feedback on code. These contrasts jointly change every active component and do not isolate a single component implementation. The direction and statistical conclusion also vary across models, domains, and endpoints. Near-call-budget controls, repeated runs, and a 500-task MATH-500 validation show that these treatments are implementation bundles whose effects include inference cost and operational reliability. On two local models, 3,200 planning episodes show no clear accuracy recovery after summary truncation is removed, despite substantially higher input-token costs. An isolated 1,600-condition feedback experiment shares first answers across revision prompts: self-checking outperforms a same-call neutral revision on Llama but not Qwen. HarnessBench therefore supports implementation-level diagnosis; two implementations per component do not estimate population-level implementation variance or identify a universal best harness.

1 INTRODUCTION

LLM agents rely on runtime infrastructure that prepares context, retains state, checks outputs, and decomposes tasks (Lopopolo, 2026; Bui, 2026). We use *harness* as an operational term for this infrastructure while recognizing the overlap with *workflow*, *pipeline*, and *scaffold*. Existing work searches over whole harnesses or studies fixed workflows (Lee et al., 2026; Pan et al., 2026); those results motivate component-level diagnosis but do not imply that an abstract label such as "planning" has a stable effect.

We separate bundle comparisons, isolated feedback contrasts from shared first answers, and cost/failure audits. Bundle effects do not identify a single component's causal effect.

HarnessBench evaluates two global implementation variants, each with a full factorial over four components and a shared null. Switching variants changes every enabled implementation; it is not a one-component replacement. Within each variant, the design asks how enabling a component changes performance across the other component settings. Our contributions are:

1. a modular 2^4 factorial framework for Context, Memory, Feedback, and Planning, with two active implementations and a common null;
2. a 4,608-evaluation, complete-output experiment crossing three model identifiers, two domains, 16 configurations, and both variants, with paired implementation contrasts;
3. controls for inference-call budget, temperature-zero repeatability, and a pinned MATH-500 validation; and
4. local-model controls with 3,200 planning episodes and 1,600 shared-first-answer feedback conditions, separating summary presentation and revision wording from broader bundle comparisons.

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.

Table 1: Dual-implementation design. Variant A favors engineered heuristics; Variant B favors model-native capabilities. Both share a common null (OFF) baseline.

Component	Variant A (feature-rich)	Variant B (minimal/alternative)	Null (OFF)
Planning	Multi-step decomposition (2–5 substeps) with progress tracking	Chain-of-thought single prompt	Single prompt, no decomposition
Feedback	Rule-based format checks + 2 retries	LLM self-check + 1 retry	No verification, accept first output
Context	8,192-token estimate, conditional compression, history compression	Structured prompt template, full history kept	Full history, no compression
Memory	Importance-weighted episodic store (50 entries, top-k retrieval)	Simple sliding window (10 recent entries)	No persistence between steps

2 Related Work

Harness engineering and optimization. OpenAI (Lopopolo, 2026), Böckeler (2026), and Bui (2026) describe harnesses and scaffolds as runtime systems around a model. Meta-Harness (Lee et al., 2026) and ACE (Zhang et al., 2026) optimize broader harness configurations. Their reported gains concern a broader search setting and are not direct baselines for our fixed-component ablations.

Agent evaluation. HAL (Kapoor et al., 2026) evaluates across models, benchmarks, and scaffolds; VeRO (Ursekar et al., 2026) provides scaffold-agnostic evaluation; SkillsBench (Li et al., 2026) benchmarks agent skills; and HarnessCard (He et al., 2026) proposes standardized reporting.

Factorial design in ML. Our approach draws on factorial experimental design (Montgomery, 2017; Box et al., 2005) and Shapley attribution (Shapley, 1953). ShapleyFlow evaluates all 2^4 model-upgrade coalitions in a fixed four-component workflow and introduces CapaBench (Yang et al., 2026); AgentSHAP attributes agent performance to tool subsets (Horovicz, 2025). HarnessBench is complementary: it compares two concrete runtime implementations per component against a shared null. With only two implementations, however, it does not estimate how implementations are distributed in the broader design space.

3 The HarnessBench Framework

3.1 Harness Component Taxonomy

We study four runtime components. **Context (C)** prepares history and retrieved memory; **Memory (M)** stores information between steps; **Feedback (F)** checks an answer and may retry; and **Planning (P)** controls whether a task is executed directly or decomposed. The 2^4 design evaluates every coalition. An OFF component is a pass-through shared by Variants A and B.

The full factorial contains $2^4{=}16$ configurations per variant. It allows us to average each ON–OFF comparison over all eight settings of the other components (Montgomery, 2017; Box et al., 2005). Each OFF implementation is a pass-through. The resulting contrast concerns the component as implemented, including its calls, prompts, and interactions with the rest of the harness.

3.2 Dual-Implementation Design

To examine sensitivity to concrete implementations, we designed two implementations per factor; this does not identify a population-level effect of a component concept. Table 1 summarizes the design.

The two variants embody different design philosophies rather than a "strong vs. weak" ranking. Variant A relies on engineered heuristics: rule-based format checks, keyword-overlap filtering, importance-weighted retrieval, and explicit multi-step decomposition with progress tracking. Variant B leans on model-native capabilities: an LLM self-check for feedback, chain-of-thought prompting in a single call for planning, a structured prompt template (no compression) for context, and a simple sliding window for memory. This contrast tests the sensitivity of a bundled operational configuration; it does not identify an abstract concept effect.

Implementation specifics. For *Context (A)*, history compression activates only above six entries and retains the first two and last four. When memory is present, the task and each memory item are

Algorithm 1 HarnessBench Agent Loop

Require: Task t, Config $\mathbf{c} = (c_C, c_M, c_F, c_P) \in \{0,1\}^4$, Model $\mathcal{M}$
Initialize components based on $\mathbf{c}$
plan $\leftarrow$ Planner.create_plan($t, \mathcal{M}$)
history $\leftarrow$ []
for each step s in plan **do**
  prompt $\leftarrow$ Planner.get_step_prompt(plan, t)
  ctx $\leftarrow$ ContextMgr.prepare(prompt, history, Memory.items)
  output $\leftarrow$ $\mathcal{M}$.generate(ctx.messages)
  output $\leftarrow$ Feedback.verify_and_correct(t, output, $\mathcal{M}$)
  Memory.store(s, output)
  history.append(output)
  plan.advance(output)
end for
return final output

lowercased and whitespace-split; items are ranked by word-set-intersection cardinality and the top five are retained. Its context estimate is 8,192 tokens; 4,096 is the generation-output cap, not the context limit. *Context (B)* adds a structured prompt and retains full history. *Memory (A)* combines a five-entry working buffer with an importance-weighted episodic store of up to 50 entries; *Memory (B)* is a ten-entry sliding window. *Feedback (A)* applies rule-based checks with at most two retries, whereas *Feedback (B)* uses an LLM self-check and at most one retry. *Planning (A)* first prompts, "Decompose the following task into 2–5 clear, actionable steps. Output ONLY the steps, one per line, numbered 1–5. No other text," then executes each parsed step in a separate call; each intermediate result is capped at 200 characters when passed forward. *Planning (B)* uses one call with, "Solve the following task step by step. Think through each step carefully before giving your final answer." These are complete implementation bundles, not isolated mechanisms.

3.3 Modular Architecture

Algorithm 1 presents the agent loop. Each component implements a common interface; toggling a component between its active and null implementations requires no changes to the rest of the pipeline.

4 Experimental Setup

Unified factorial. We run both variants for all 16 configurations on HB-Math (30 tasks) and HB-Code (18 tasks) using the fixed OpenRouter model identifiers `openai/gpt-4o-2024-11-20`, `deepseek/deepseek-chat-v3.1`, and `anthropic/claude-sonnet-4.6`. This yields $2 \times 16 \times 3 \times (30 + 18) = 4{,}608$ completed evaluations. The runner preserves the complete model response before evaluation; no record is empty or error-prefixed. Model identifiers were fixed, but OpenRouter's upstream provider routing used its default policy and was not pinned. Gemini 2.5 Flash was attempted but excluded before analysis because the evaluator did not reliably parse its response format; it contributes no record or performance claim.

HB-Math is a hand-curated diagnostic set of canonical exercises and is not contamination-controlled; paired comparisons reduce task-composition differences but cannot remove possible pretraining overlap. HB-Code uses executable Python tests. We additionally evaluate the three Planning conditions on the 500-task MATH-500 benchmark (Hendrycks et al., 2021), scored with `math-verify==0.9.0`.

Paired estimands. Let $Y_{tmv}(c, z)$ be the retained score for task t, fixed model m, global implementation variant v, the component setting $c \in \{0,1\}$, and the remaining three-component vector $z \in \{0,1\}^3$. The marginal component contrast for one task is

$$d_{tmv} = \frac{1}{8} \sum_{z} \{Y_{tmv}(1, z) - Y_{tmv}(0, z)\}.$$

Averaging over the fixed models before resampling tasks preserves the task as the unit of uncertainty. The implementation contrast is the corresponding average of $d_{tmB} - d_{tmA}$. Since switching v changes all active implementations, this difference includes changes in the coalitions over which the component is averaged. It is a reproducible contrast of complete harness configurations; it does not identify a context-free effect of the word "planning" or "feedback".

The isolated-component contrast instead fixes $z = (0, 0, 0)$. It answers a different question: what changes when this implementation is added to an otherwise null harness? Disagreement between marginal and isolated contrasts can arise from interactions, without either calculation being erroneous. Both are conditional on the selected tasks, endpoints, evaluation contract and available outputs. Repeating calls or changing a prompt creates a new intervention, even if both interventions carry the same component name.

Uncertainty and multiplicity. For each implementation, a main effect is the paired task-level mean difference between configurations with a component ON and OFF, averaging over the other factors and models. Direct implementation contrasts test the difference between the Variant-B and Variant-A marginal effects. We use 5,000 task-cluster bootstrap samples for 95% confidence intervals and 5,000 paired sign-flip permutations for p-values. Within each domain, four component tests use a Bonferroni threshold of $0.05/4 = 0.0125$, separately for each variant's main effects and for the direct B–A contrasts. This is not joint error control across variants, domains, model-specific analyses, and follow-up controls. The 95% intervals are pointwise, not simultaneous; the replication and follow-up analyses are diagnostic, not claimed to be preregistered. "Not distinguishable from zero" refers to this diagnostic design and resolution; it is not evidence of no effect.

Record coverage and statistical checks. The offline audit verifies all 4,608 records, including 288 complete task–model–variant cells with all 16 component vectors and no duplicate or missing vector. It reproduces the paired implementation contrasts from retained scores without executing generated code or claiming new evaluator validation. The three models are fixed in the analysis; tasks, not model–task rows, are resampling clusters. Sign-flip inference assumes symmetric/exchangeable contrast signs under the null, rather than following automatically from factorial labels. The 5,000-draw Monte Carlo test has minimum reported $p = 1/5001$; a rounded $p = .0002$ is a simulation-resolution limit, not an exact tiny tail probability. Nonempty output and no API error do not guarantee that every planned step completed or that the final answer satisfies the task.

5 Results

5.1 Factorial Effects

Table 2: Pooled main effects in the complete-output factorial replication. Each entry is effect [95% CI]; bold entries survive the four-component Bonferroni threshold.

Domain	Component	Variant A	Variant B
Math	Context	$-.035[-.063, -.008]$	$.000[-.008, .008]$
	Memory	$-.007[-.026, .011]$	$-.003[-.007, .000]$
	Feedback	$.001[-.014, .018]$	$-.028[-.065, -.003]$
	Planning	$\mathbf{-.193}[\mathbf{-.265}, \mathbf{-.122}]$	$-.006[-.036, .021]$
Code	Context	$-.054[-.099, -.008]$	$.026[-.007, .062]$
	Memory	$.028[-.012, .072]$	$-.012[-.036, .011]$
	Feedback	$-.023[-.053, .008]$	$\mathbf{.296}[\mathbf{.245}, \mathbf{.347}]$
	Planning	$\mathbf{-.207}[\mathbf{-.288}, \mathbf{-.121}]$	$-.035[-.084, .019]$

Planning-A is negative in both domains ($p = .0002$ on Math and $p = .0006$ on Code), while Feedback-B is positive on Code ($p = .0002$). The other pooled effects do not cross the corrected threshold. Feedback-B's Code effect is positive for all three model identifiers: GPT-4o $+.116$ ($p = .0078$), DeepSeek $+.086$ ($p = .0046$), and Claude $+.688$ ($p = .0002$). Claude's magnitude reflects a low first-pass executable-code score and a corrective second call, so it should not be generalized beyond this setup.

5.2 Direct Implementation Contrasts

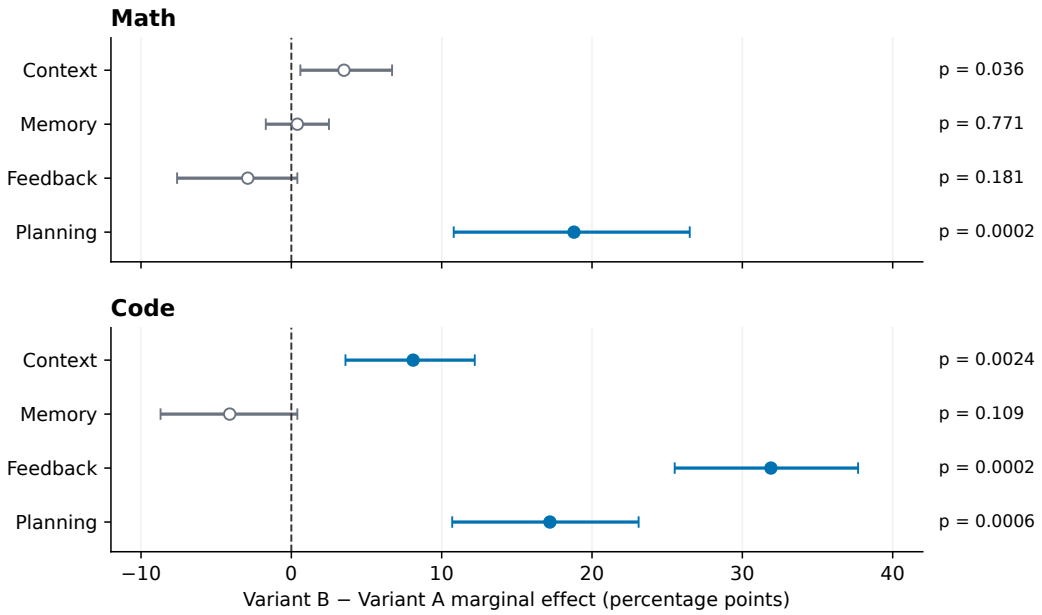


Figure 1: Paired Variant-B minus Variant-A marginal effects, in percentage points. Lines are pointwise 95% task-cluster bootstrap intervals; filled blue markers pass the four-component Bonferroni threshold $p < .0125$, while hollow markers do not. Monte Carlo sign-flip p-values are printed alongside. Correction is applied separately within each domain, not jointly across all analyses.

Changing the concrete implementation materially changes the estimated effect for Planning in both domains and for Context and Feedback on Code. This supports implementation-dependent effects in these tested pairs. It does not establish that implementation "matters more" than component choice: the experiment defines no population distribution over possible implementations from which such a variance comparison could be estimated.

Reading the implementation contrast. The positive Planning B–A contrast should not be read as evidence that adding Planning-B improves over the null harness. It can arise because Planning-A is substantially worse: the separate main-effect table shows B close to zero in both domains. Likewise, a pointwise interval excluding zero is not the same as passing the corrected component test. Math Context illustrates this distinction: its interval is positive, but its reported $p = .036$ exceeds the four-test threshold. Reporting the two estimands and the decision rule together avoids turning a relative implementation comparison into an unsupported claim about the benefit of a component.

The Code Feedback contrast is larger, yet the additional neutral-revision experiment asks a stricter question. It shares the first answer, call allowance and available prefix across the two second-call prompts. Qwen gains no clear advantage from self-check wording beyond neutral revision, whereas Llama does. The factorial and isolated experiments use different models, tasks and interventions. Their contrast is informative: a positive feedback effect against no revision need not imply a benefit from self-check wording over another revision call.

5.3 Cost, Repeatability, and External Validation

Planning-A uses several calls while Planning-B and the bare condition use one, so their marginal contrast mixes mechanism, compute, and failure exposure. On DeepSeek-V3.1 HB-Math, a six-call CoT self-consistency control uses the majority of parsed final answers, breaking ties by the earliest candidate. It scores .967 at 6.00 calls and 2,422 tokens per task. Three Planning-A runs score .867/.633/.733 (mean .744) at 6.17 calls and 3,603 tokens; the paired self-consistency-minus-Planning-A difference is $+.222$ $[.133, .322]$, $p = .0002$. This near-call-budget control does not

Table 4: Frozen local-model controls: accuracy (percent), 200 tasks per cell. A-trunc and A-full share the same initial plan. These are concrete implementation comparisons, not effects of planning in general.

Domain	Model	Bare	B	A-trunc	A-full
MATH	Qwen	78.5	76.0	65.0	65.0
MATH	Llama	54.5	51.0	40.5	41.5
MBPP	Qwen	52.5	49.5	39.0	35.5
MBPP	Llama	38.0	25.0	24.0	24.0

recover a Planning-A advantage; calls are not exactly matched and token budgets differ, although this remains a comparison of two complete algorithms rather than a pure decomposition effect.

Table 3: Temperature-zero DeepSeek-V3.1 repeats on HB-Math. Values after $\pm$ are sample standard deviations across three runs.

Condition	Accuracy	Calls/task	Tokens/task
Bare	$.900 \pm .000$	1.00	338
Planning-A	$.744 \pm .117$	6.17	3,603
Planning-B	$.911 \pm .019$	1.00	413
Feedback-A	$.900 \pm .000$	1.00	293
Feedback-B	$.900 \pm .000$	2.00	592

Planning-A varies substantially across nominal temperature-zero runs, whereas the other tested conditions vary by at most one HB-Math item. Because upstream routing was not pinned, this repeatability result captures the combined operational variation of the endpoint and routing rather than model-internal nondeterminism alone. One-run temperature sensitivity checks at $0.1/0.5/0.7$ give Planning-A $.633/.867/.800$ and Planning-B $.967/.900/.900$; these are descriptive because temperature and default upstream routing are not separately identified.

On MATH-500, all 1,500 attempts have nonempty outputs. Bare, Planning-A, and Planning-B score $.948$, $.734$, and $.960$. Relative to bare, Planning-A is $-.214$ $[-.254, -.174]$, $p = .0002$, using 5.75 calls, 1.76 completed steps, and 6,764 tokens per task; it is lower in every subject and difficulty group. Planning-B is $+.012$ $[-.008, .032]$, $p = .326$, with one call and 988 tokens. These results contradict a universal claim that Planning-A helps mathematics and instead expose the operational reliability of this particular multi-step bundle.

5.4 Does removing summary truncation recover accuracy?

We add 3,200 completed episodes using local Qwen2.5-7B-Instruct and Llama-3.1-8B-Instruct, 200 fixed MATH-500 tasks and 200 fixed official MBPP test tasks, and four conditions: bare, Planning-B, Planning-A with truncated summaries, and Planning-A with full summaries. Task IDs and interfaces were frozen before running; engineering tasks are excluded. Decoding is greedy, with at most six calls and 8,192 completion tokens per episode. Both A conditions reuse exactly the same initial plan for a task; each deployment comparison is charged its own planning call, while actual shared GPU occupation is counted once.

The truncated condition limits displayed intermediate summaries, but the full original conversation remains available. This is a presentation/repetition intervention, not deletion of all access to previous information. Incorrect answers, invalid plans, output/context limits and archived execution failures remain in the denominator. The MBPP primary analysis retains all 200 tasks, including one official reference with a failing floating-point equality assertion; a prespecified 199-task sensitivity is also reported.

Compared with bare inference, A-trunc loses 13.5–14.0 percentage points and A-full loses 13.0–17.0 points across the four model/domain cells. All eight pointwise paired-task bootstrap intervals are below zero. In contrast, A-full minus A-trunc is 0.0 points $[-3.5, 3.5]$ for Qwen/MATH, $+1.0$ $[-4.5, 6.0]$ for Llama/MATH, -3.5 $[-9.0, 1.5]$ for Qwen/MBPP, and 0.0 $[-5.0, 5.0]$ for Llama/MBPP.

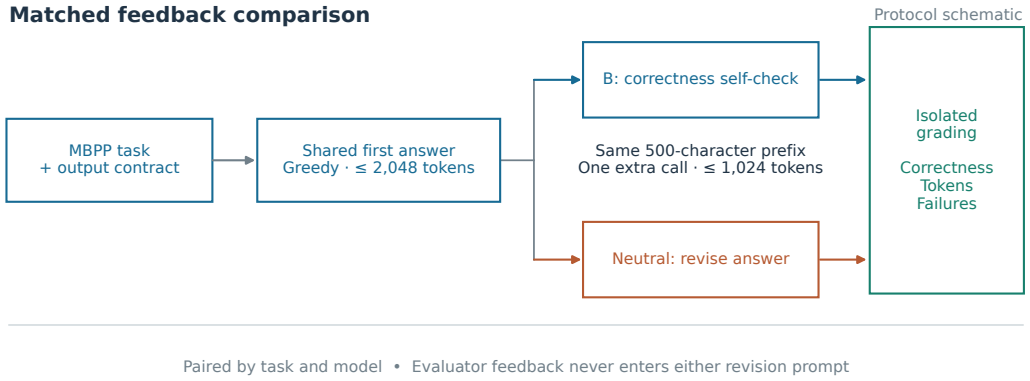


Figure 2: Matched B-versus-neutral feedback comparison. Both revision prompts receive the same saved first-answer prefix and one extra call; only the revision wording changes. Evaluator feedback is withheld. Correctness, token costs and incomplete outputs are assessed separately. The Null and rule-based A arms are omitted here; this comparison differs from the global A/B bundle contrasts.

The intervals use 10,000 task-resampling draws (seed 180914) and are pointwise, without a joint multiplicity guarantee. The 199-task MBPP sensitivity retains the same qualitative conclusions. Thus simply removing these summary limits does not demonstrate recovery of Planning-A performance; the intervals do not establish equivalence either.

A-full averages 18,407–25,605 input tokens per episode across cells, compared with 8,635–10,886 for A-trunc. Both use approximately 5.5–6.0 calls, while bare and B each use one. In Llama/MBPP, A-full has 27 incomplete episodes versus four for A-trunc, despite both scoring 24%. Full-summary context pressure and output errors are part of the implementation bundle and cannot alone identify a causal failure mechanism. Costs include successfully returned input/output tokens; rejected context requests have no generated-token charge. Shared-service queueing contaminates wall-clock latency, so we do not use it to claim a pure inference speedup.

The analysis includes all 3,200 episodes, with failure-inclusive grades and call and token costs computed from their responses. Reusing these grades does not constitute independent semantic regrading or prove that the tests are exhaustive. One greedy run per task supplies task-level uncertainty only, not variability over sampling seeds or over the population of possible harness implementations.

6 Does self-checking help beyond another revision call?

We first replay all 1,600 saved MBPP outputs from the prior local-model experiment under the original tests in an OS-isolated evaluator; every label agrees. A conservative MBPP+ v0.2.0 (EvalPlus contributors, 2024) join requires identical normalized task text and public parameter signatures, followed by compatibility of the original reference with the extended inputs. Of 79 overlapping IDs, 25 satisfy this protocol (46 text mismatches, three signature mismatches, and five reference-compatibility or resource-limit exclusions). This is a conservative differential-test subset with whole-program time limits, not the official EvalPlus leaderboard metric. On these 25 tasks, original/extended correct counts for bare, Planning-B, truncated Planning-A, and full Planning-A are respectively 14/14, 16/14, 11/10, 10/9 for Qwen, and 8/7, 10/10, 7/7, 7/7 for Llama. We retain every exclusion and all outputs.

The new feedback experiment selects 200 official MBPP test tasks by frozen hash, excluding the prior 200 tasks and engineering examples. Context, memory, and planning are OFF. Each model shares one greedy first answer across Null, the original rule-based Feedback-A (up to two retries), the original self-check Feedback-B (one retry), and a neutral revision prompt. The neutral comparator uses the same one-call allowance and 500-character first-answer prefix as B, replacing only its correctness-check request. First answers permit 2,048 new tokens; revisions permit 1,024. No evaluator feedback is exposed to a model. Feedback-A retains its original math-keyword/numerical-answer heuristic

even on coding tasks; it is not an execution verifier. With 10,000 paired task-bootstrap samples, B

Table 5: Isolated feedback: accuracy on 200 new MBPP tasks per model. All four arms share the first answer.

Model	Null	Feedback-A	Feedback-B	Neutral revision
Qwen2.5-7B-Instruct	50.0%	49.5%	54.5%	55.0%
Llama-3.1-8B-Instruct	30.0%	29.0%	41.5%	25.5%

minus neutral revision is -0.5 percentage points (95% CI $[-4.0, 2.5]$) for Qwen and $+16.0$ points ($[10.0, 22.5]$) for Llama. Thus the self-check wording has model-dependent incremental value beyond one ordinary revision call. B and neutral each use two total calls; A averages 1.25 and 1.28. Mean input tokens increase from 69.6/75.6 under Null to 296.6/297.6 under B, and completion costs also differ. These are operational prompt interventions on two models and one coding benchmark, not evidence for universal feedback effectiveness.

7 Output-contract implementation audit

The historical feedback implementation inferred a mathematical task from words such as "find" and "compute", including when the benchmark requested code. It also returned `verified=True` after exhausting retries without checking whether the last output satisfied the checker. The revised checker receives an explicit task type, language, and output contract. For Python it checks nonempty output, syntactic validity, and the requested entry signature; these checks do not establish semantic correctness and do not access hidden tests. Verification after a retry is derived from the final contract check, with failed calls counted against the call budget. The historical implementation and its results are retained. These changes specify the corrected contract; they do not establish its benchmark accuracy. The historical performance measurements apply to the original checker and cannot be reassigned to the corrected implementation. The title distinguishes this focused audit from the separate Harness-Bench study (`https://arxiv.org/abs/2605.27922`).

This distinction also determines the denominator. A failed call spends resources even when it yields no usable answer; an exhausted retry cannot be called verified merely because the retry loop terminates. A syntactically valid program can still fail hidden tests. Consequently, operational completion, output-contract validity and task correctness are separate outcomes. The corrected checker addresses the first two, while a semantic evaluator supplies the third. The correction does not retroactively change the saved historical intervention or establish its performance under a newly corrected checker.

8 Discussion

The evidence suggests four reporting practices. First, name and ablate concrete implementations rather than assigning an effect to an abstract component label. Second, report model, domain, evaluator, and provider scope because sign and significance can change across them. Third, report quality jointly with calls, tokens, and failed or incomplete steps. Fourth, do not attribute a bundled effect to one internal mechanism without a targeted control. Context-A uses an 8,192-token context estimate and a 4,096-token output cap; its filtering and compression are conditional. Its effect therefore cannot be attributed to a 4K context cap.

Identification under global variants. The original variant switch changes all active component implementations together. Differences between marginal effects averaged over non-null coalitions can therefore include changed interactions with other components. We additionally estimate all 24 model/domain/component contrasts with every other component OFF, using the 4,608 saved records (30 math and 18 code tasks, one observation per cell). These isolate the chosen operational component under an otherwise null harness, but still bundle prompt changes, call count and failure exposure. They are not interchangeable with the global marginal effects.

Limitations. Two implementations are examples, not a sample sufficient to characterize an implementation population. HB-Math and HB-Code are small diagnostic sets, and HB-Math is not

Table 6: Interpretation of controls used in the audit. A control fixes only the quantities shown; it does not remove the remaining limitation.

Comparison	Held fixed	Remaining limitation
Global factorial	Tasks, models, component vectors	Active implementations and interactions differ
Isolated component	Other three components OFF	Prompt, calls and failure exposure can differ
B versus neutral revision	First answer, extra call, prefix allowance	Revision wording and completion length differ
Contract regression	Authored input and expected contract	No semantic benchmark accuracy is established

contamination-controlled. API models can vary behind fixed identifiers; upstream routing was not pinned. Bootstrap intervals capture task variation, while the dedicated repeats reveal run-to-run variation only for selected conditions. The active treatments are bundles that change prompts, call count, state propagation, and failure exposure, so factorial effects should not be interpreted as pure causal effects of a single internal mechanism. Our four-component taxonomy also omits tool dispatch, safety controls, and multi-agent coordination.

Interpreting a deployment decision. The matched revision comparison answers a narrower question than whether an agent should use feedback. Its baseline already spends an extra model call, so the B-minus-neutral contrast measures the incremental effect of the chosen self-check request at that allowance. A B-minus-Null contrast also includes the opportunity to revise. These are different decisions: whether to allocate another call, and how to use that call once allocated. The Llama result supports the self-check wording under the tested contract; the Qwen interval leaves its incremental effect unresolved. Neither result identifies a prompt that maximizes accuracy at a fixed token budget, because output lengths differ.

For the same reason, a resource budget should be specified before selecting among implementations. Equal call counts do not imply equal context length, generation length or failure probability. Counting only successful outputs would condition on an outcome changed by the treatment. The reported failure-inclusive denominator instead evaluates the implemented procedure, including occasions when it fails to return a usable answer. A corrected output contract changes that procedure and therefore requires its own benchmark evaluation; engineering regression tests cannot supply the missing accuracy estimate.

From a diagnostic to a harness choice. A component's average factorial effect is not a prescription for a particular harness. The average covers multiple coalitions; deployment fixes one combination, where interactions may change the contrast. Isolated-component estimates instead describe a null-harness starting point. Both views can narrow the candidate set, but selecting a complete harness requires a direct comparison on the intended tasks under a common resource allowance. Failed calls and incomplete steps remain outcomes, and context carried into later calls contributes to cost. The shared-first-answer experiment illustrates a focused comparison: it separates revision wording from the opportunity to obtain another answer, while leaving token-budget optimality unresolved.

9 Conclusion

HarnessBench provides a controlled way to diagnose concrete harness implementations around a fixed model. In the complete-output 4,608-evaluation replication, paired A/B contrasts differ for Planning on both domains and for Context and Feedback on Code. Repeats, a near-call-budget control, and MATH-500 show why implementation cost and reliability must accompany accuracy. The appropriate conclusion is conditional: component effects depend on the tested implementation, model, domain, and endpoint; no universal component ranking or best harness follows from these experiments.

Reproducibility Statement

The methods state the factorial configurations, model identifiers, task sets, implementation contrasts, and paired resampling procedures. Retained evaluation records and audit scripts support the complete-output replication and its contrast analysis. The follow-up protocols specify frozen task selection, shared first answers, revision budgets, and isolated execution checks; the feedback diagram depicts the matched B-versus-neutral comparison. Historical outputs and newly executed controls are kept distinct, and saved-record agreement does not independently validate an evaluator or eliminate model-version dependence.

AI Use Statement

OpenAI Codex was used to aid and polish manuscript writing and to assist with literature retrieval and discovery, including finding related work. The authors are responsible for the final manuscript and citations.

References

Birgitta Böckeler. Harness engineering for coding agent users. https://martinfowler.com/articles/exploring-gen-ai/harness-engineering.html, 2026. Accessed: 2026-04-01.

George E. P. Box, J. Stuart Hunter, and William G. Hunter. *Statistics for Experimenters: Design, Innovation, and Discovery*. Wiley, 2nd edition, 2005.

Nghi D. Q. Bui. Building effective AI coding agents for the terminal: Scaffolding, harness, context engineering, and lessons learned. *arXiv preprint arXiv:2603.05344*, 2026.

EvalPlus contributors. MBPP+ dataset, version 0.2.0. GitHub release, 2024. URL https://github.com/evalplus/mbppplus_release/releases/tag/v0.2.0. Frozen release archive and evaluator source hashes included in the artifact.

Chaoyue He, Xin Zhou, Di Wang, Hong Xu, Wei Liu, and Chunyan Miao. Harness engineering for language agents: The harness layer as control, agency, and runtime. *Preprints.org, manuscript 202603.1756*, 2026.

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. *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2021.

Miriam Horovicz. AgentSHAP: Interpreting LLM agent tool importance with monte carlo shapley value estimation. *arXiv preprint arXiv:2512.12597*, 2025. URL https://arxiv.org/abs/2512.12597.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S. Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, et al. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/a0928f924a344aaebbb7f6cd8d56e34c-Abstract-Conference.html.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026.

Ryan Lopopolo. Harness engineering: leveraging Codex in an agent-first world. https://openai.com/index/harness-engineering/, 2026. OpenAI Blog, February 11, 2026. Accessed: 2026-04-01.

Douglas C. Montgomery. *Design and Analysis of Experiments*. Wiley, 9th edition, 2017.

Linyue Pan, Lexiao Zou, Shuo Guo, Jingchen Ni, and Hai-Tao Zheng. Natural-language agent harnesses. *arXiv preprint arXiv:2603.25723*, 2026.

Lloyd S. Shapley. A value for n-person games. In Harold W. Kuhn and Albert W. Tucker (eds.), *Contributions to the Theory of Games II*, pp. 307–317. Princeton University Press, 1953.

Varun Ursekar, Apaar Shanker, Veronica Chatrath, Yuan Xue, and Samuel Marc Denton. VeRO: A harness for agents to optimize agents. In *Proceedings of the International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/60518.

Yingxuan Yang, Bo Huang, Siyuan Qi, Chao Feng, Haoyi Hu, Yuxuan Zhu, Jinbo Hu, Haoran Zhao, Ziyi He, Xiao Liu, ZongYu Wang, Muning Wen, Lin Qiu, Xuezhi Cao, Xunliang Cai, Yong Yu, and Weinan Zhang. Attribution-based analysis and optimization of modular agentic workflows. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 7247–7264. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.359. URL https://aclanthology.org/2026.findings-acl.359/.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8a94ff6f922d995d7d3f4ebf4143e442-Abstract-Conference.html.