DELETE OR PRESERVE? AN EMPIRICAL STUDY OF THE EFFECTS OF DELETION AND PRESERVATION INSTRUCTIONS ON AI-GENERATED CODE

Anonymous authors
Paper under double-blind review

ABSTRACT

The two leading AI coding tools lean in opposite directions on removing code: Claude Code's system prompt says that if the model is "*certain that something is unused, you can delete it completely*," while OpenAI's Codex guide says to "*NEVER revert existing changes you did not make unless explicitly requested*." Neither vendor has published measurements of what such guidance does. We generalize each into an explicit instruction—one to delete dead code and prefer subtractive fixes, one to preserve existing code and fix additively—and append it to the request as a controlled manipulation across five LLMs (3,600 runs) on 24 code-repair tasks whose correct fix removes code. **The preservation instruction lowers correctness on every model**: by 13–15 percentage points on Claude Sonnet 4, Gemini 2.5 Pro, and Qwen3-32B, and by 39 points on GPT-5-mini. With tasks rather than runs as the unit of analysis, the drop remains significant on GPT-5-mini and Qwen3-32B and is borderline on Gemini; on Claude it comes from four of the 24 tasks. The deletion instruction never harms correctness. A failure-mode analysis shows that wrapping is the baseline failure on every model, while preservation mainly amplifies downstream patching. In 57,371 deployed commits from the AIDev dataset, Codex has the lowest deletion ratio, as the mechanism predicts; Claude Code also deletes below the human rate despite its pro-deletion prompt, suggesting that base-model reluctance to delete persists against instruction. A pre-registered follow-up tests six scope-limited rewordings of the preservation instruction on three local models: all recover most of the lost correctness, but each misses the pre-registered over-deletion criterion on Llama 4, so none clears the gate. The instructions coding tools give their models are engineering artifacts whose costs can, and should, be measured.

1 INTRODUCTION

A system prompt is a short block of instructions that an AI coding tool prepends to every model invocation. It is fixed by the tool vendor, invisible to the developer, and treated as scaffolding rather than software. Yet it ships in production to millions of developers and is, in a meaningful sense, the most-executed code path in the tool. The two largest deployments lean in opposite directions on a question that matters for software maintenance: when to remove code. Anthropic's Claude Code system prompt tells the model Anthropic (2026): "*Avoid backwards-compatibility hacks like renaming unused _vars, re-exporting types, adding // removed comments for removed code, etc. If you are certain that something is unused, you can delete it completely.*" OpenAI's Codex prompting guide, under the heading "You may be in a dirty git worktree," tells it OpenAI (2025): "*NEVER revert existing changes you did not make unless explicitly requested, since these changes were made by the user.*" The two target different situations—dead code on one side, a user's uncommitted edits on the other—so they do not literally contradict each other. But one licenses removal and the other guards against it, and neither vendor has published evidence about what either kind of guidance does to the code a model produces. This paper measures that directly, using explicit, general-purpose versions of each.

Industry reports motivate the question: refactoring rates in AI-assisted codebases have dropped from 25% to under 10% of changes GitClear (2024; 2025), code duplication has risen eightfold GitClear

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.

(2024; 2025); SonarSource (2025), and pull-request size has grown 154% Google DORA Team (2025). Humans systematically overlook subtractive changes Adams et al. (2021), and LLMs inherit and sometimes amplify this addition bias Uhler et al. (2026); Santagata & De Nobili (2025). We call the code-editing form of this tendency *shyness*: an LLM's reluctance to delete or simplify existing code, defaulting instead to additive edits—wrapping, duplicating, or commenting out—that leave the original in place. Our benchmark (*ShyBench*) and the *shyness rate* metric (Section 3) are named for it. What is missing is a controlled measurement of how much of this behavior is induced by the instructions a tool gives the model versus baked into the model.

We test two contrasting instructions generalized from this guidance (quoted in full in Section 3) as the manipulation in a controlled experiment across five LLMs spanning five model families and four providers (Claude Sonnet 4, Gemini 2.5 Pro, Qwen3-32B, Gemma 4-31B, and GPT-5-mini Anthropic (2025); Google DeepMind (2025); Qwen Team, Alibaba Cloud (2025); Google DeepMind (2026)) on 24 code-repair tasks, each constructed so the gold fix requires deletion or simplification. **The preservation instruction degrades code correctness on every model** and by 13–15 percentage points on three of five, surviving Holm correction across a 30-pairwise-test grid. On GPT-5-mini, the only OpenAI model in our panel, the drop is 39 pp. With tasks rather than runs as the unit of analysis, the drop remains significant on GPT-5-mini and Qwen3 and is borderline on Gemini; Claude's comes from four of the 24 tasks. The deletion instruction is neutral-to-helpful and never harms correctness. An observational re-analysis of 57,371 deployed commits is consistent with this mechanism, and a pre-registered follow-up finds that no scope-limited rewording of the preservation instruction we tested clears its pre-registered gate.

Contributions.

- A **controlled measurement of deletion and preservation instructions** across five LLMs and three prompting conditions (3,600 runs). The headline is a *correctness* effect, not a stylistic one: preservation reduces correct-patch rates by 13–15 pp on three of five models and by 39 pp on GPT-5-mini; deletion is neutral-to-positive on every model (Section 4).
- A **failure-mode diagnosis** using an eight-category additive taxonomy: wrapping (B1) is a baseline behavior on all five models; preservation amplifies downstream patching (B7), or duplication (B3) and abstraction inflation (B5) on GPT-5-mini (Section 4, Appendix A).
- An **observational consistency check** on 57,371 deployed commits: Codex, whose guidance leans toward preservation, has the lowest deletion ratio in production use, and Claude Code deletes below the human rate despite a prompt that licenses deletion (Section 5).
- A **pre-registered mitigation experiment** (Phase 2) testing six scope-limited variants and per-model targeted prompts on three local models: each recovers most of the lost correctness, but none is shown to keep over-deletion as low as blanket preservation, so none clears the gate (Section 6).
- A **released artifact** (24 controlled tasks, 12 benefit-measuring tasks, ≈8,600 archived runs, classifiers, and statistical pipelines) that reproduces every table and figure from raw outputs.

2 Related Work

Addition bias in humans and LLMs. Adams et al. Adams et al. (2021) demonstrated in *Nature* that humans systematically overlook subtractive changes across eight experiments. Two recent papers extend this to LLMs. Santagata and De Nobili Santagata & De Nobili (2025) tested six models and found all showed significant addition bias: Llama 3.1 chose addition 97.85% of the time in a palindrome task. Uhler et al. Uhler et al. (2026), published in *Nature Communications Psychology*, found that addition bias is *more pronounced* in GPT-4 than in humans. Neither paper studied the bias in a code editing context—the gap our work fills.

LLM sycophancy and prompt sensitivity. Sharma et al. Sharma et al. (2023) showed that RLHF training encourages models to match user preferences over truthful responses. Broader evaluations Fanous et al. (2025); Cheung et al. (2025); Echterhoff et al. (2025) confirm that alignment training amplifies systematic biases, and that those biases are sensitive to instruction-level context. The preservation-style instructions that we study can be read as one such context: an instruction

to defer to the user's prior code as if it were a stated preference. Our results extend the prompt-sensitivity literature into the code-editing setting and show that a short instruction can move correctness by 13–15 percentage points.

System prompts as engineering artifacts. A complementary line of work treats system prompts and meta-prompts as software in their own right, with measurable effects on downstream behavior. We are not aware of prior empirical work that measures how deletion- or preservation-oriented guidance, of the kind the two largest coding-tool vendors ship in narrower forms, affects the correctness of generated code. This paper provides that measurement and tests six scope-limited variants of the preservation instruction as candidate mitigations (Section 6).

AI-assisted code quality. GitClear GitClear (2024; 2025) analyzed 153 million changed lines and found refactoring dropped from 25% to under 10% while code duplication rose eightfold. Google's DORA report Google DORA Team (2025) found 154% larger pull requests (Faros telemetry as cited in DORA). Ogenrwot and Businge Ogenrwot & Businge (2026) found human PRs have higher median deletions than AI agents on the AIDev dataset (Cliff's $d = 0.45$). Sajadi et al. Sajadi et al. (2025) found LLM patches are larger than gold patches. Studies on refactoring Cordeiro et al. (2024) show models *can* delete when explicitly asked—indicating a behavioral default, not a capability gap. The alignment tax literature provides a mechanistic explanation: safety training degrades performance Huang et al. (2025) and can cause broader misalignment Betley et al. (2026).

3 Controlled Study Design

We construct a controlled environment in which the correct fix requires deletion, and we manipulate a short instruction appended to the request. The instructions are modeled on the vendor guidance quoted in Section 1 Anthropic (2026); OpenAI (2025), but are deliberately more explicit and more general than it.

Benchmark. ShyBench contains 24 Python tasks (50–200 lines each), three per additive failure mode B1–B8: *wrapping* instead of fixing (B1), *commenting out* instead of deleting (B2), *duplication* instead of replacement (B3), *hoarding* dead code (B4), *abstraction inflation* (B5), *configuration* over deletion (B6), *downstream patching* (B7), and *deprecation theater* (B8). We use this taxonomy as an analytical instrument, not a primary contribution; full definitions are in Appendix A. Each task has a buggy snippet, a gold subtractive fix, a plausible additive alternative, and a test suite of functional tests plus *meta-tests* that detect shy patterns (e.g., no added `try/except` for B1, no `_v2` function for B3); see Appendix B.1.

Conditions. Every run is a single user message with no system prompt. In the **baseline** condition it contains a task-specific request (e.g., "This codebase migrated from custom file-based logging to Python's logging module. Clean up the dead code."), the code, and "Return ONLY the complete fixed file as a Python code block. No explanation." The two treatment conditions append one final paragraph to that message, quoted here in full:

- **Deletion-encouraging:** "IMPORTANT: Delete unused code completely rather than adding compatibility shims. If code is dead, remove it. Prefer minimal, subtractive fixes."
- **Preservation-encouraging:** "IMPORTANT: Never revert or remove existing code unless explicitly requested. Preserve existing functions and structure. Add new code to fix issues rather than deleting existing code."

Neither is the vendor's verbatim text. The deletion instruction drops Claude Code's "if you are certain" condition; the preservation instruction widens Codex's protection of the user's uncommitted edits to all existing code, and its last sentence directly asks for additive fixes. Our results therefore measure explicit, general deletion and preservation guidance delivered in the user turn, not the shipped system prompts; the narrower shipped wording may well have a smaller effect, and we did not test it (Section 7).

Metrics. For each patch we compute *net bloat* β = lines added − lines deleted and the *deletion ratio* δ = lines deleted/(lines added+lines deleted). The test suite (functional + meta-tests) decides pass/fail; a passing patch with $\beta \leq 0$ is *subtractive*, otherwise *additive*. The **primary outcome** is the

Table 1: Headline results (24 tasks, n=10, 3,600 total runs). **Bold**: the comparison to baseline survives Holm correction at $\alpha = 0.05$. Brackets: 95% Wilson intervals. N varies because API errors and empty responses are excluded; counting them as failures preserves or strengthens every correctness effect (Appendix B.5).

Model / Cond.	N	Corr% [CI]	Shy% [CI]	Model / Cond.	N	Corr% [CI]	Shy% [CI]
Claude S4 base	240	85 [80, 89]	10 [7, 15]	Gemma 4-31B base	239	95 [91, 97]	6 [3, 10]
Claude S4 del	240	88 [84, 92]	8 [5, 13]	Gemma 4-31B del	237	95 [92, 97]	2 [1, 5]
Claude S4 pres	240	**70** [64, 76]	12 [8, 18]	Gemma 4-31B pres	236	92 [88, 95]	9 [6, 14]
Gemini 2.5 base	236	93 [89, 96]	7 [4, 11]	GPT-5-mini base	240	94 [90, 96]	13 [9, 18]
Gemini 2.5 del	229	94 [91, 97]	6 [4, 10]	GPT-5-mini del	240	**99** [97, 100]	**1** [0, 3]
Gemini 2.5 pres	209	**79** [73, 84]	13 [9, 19]	GPT-5-mini pres	240	**55** [49, 61]	**26** [19, 34]
Qwen3-32B base	236	91 [87, 94]	9 [6, 14]				
Qwen3-32B del	232	91 [87, 94]	5 [3, 9]				
Qwen3-32B pres	231	**78** [73, 83]	10 [7, 16]				

correctness rate: the fraction of runs that pass the full suite. Secondary outcomes are the **shyness rate** (fraction of *correct* patches that are additive) and **Shy**$_F$ (incorrect patches with $\delta < 0.3$ caught by meta-tests). Correctness measures the engineering harm; shyness isolates one mechanism within it.

Models and statistics. We test Claude Sonnet 4, Gemini 2.5 Pro, and GPT-5-mini via API, and Qwen3-32B and Gemma 4-31B locally via Ollama (Q4_K_M), with n=10 runs per task-condition: 720 runs per model, **3,600 total**. Claude Sonnet 4 and GPT-5-mini are the model families behind Claude Code and Codex, linking the controlled and observational studies. GPT-5-mini's API requires `temperature=1` (with `reasoning_effort='low'`); the others run at `temperature=0.7`—an asymmetry we discuss in Section 7. Each pairwise comparison is a 2×2 table tested with two-sided **Fisher's exact test**; we report **Cliff's** d for every comparison, significant or not (for a comparison "A vs. B", $d > 0$ means A has the higher rate; we write d rather than δ, which denotes the deletion ratio), and apply **Holm correction** across the full grid of 30 tests (5 models $\times$ 2 outcomes $\times$ 3 condition pairs), writing p_{Holm} for adjusted values. Benjamini–Hochberg yields the same significant set. Details are in Appendix B.3.

4 Controlled-Study Results

Finding 1: Preservation degrades correctness, robustly. The preservation instruction reduces patch correctness on every model (Table 1, Figure 1). Baseline vs. preservation: Claude -15 pp, $p_{\text{Holm}} = 0.006$, Cliff's $d = +0.142$; Gemini -14 pp, $p_{\text{Holm}} = 3 \times 10^{-4}$, $d = +0.143$; Qwen3 -13 pp, $p_{\text{Holm}} = 0.004$, $d = +0.127$; **GPT-5-mini -39 pp, $p_{\text{Holm}} < 10^{-22}$, $d = +0.388$.** Gemma 4 is resilient (-3 pp, not significant after correction). The deletion-vs.-preservation contrast is larger still and significant on the same four models ($d = +0.13$ to $+0.18$ for the three smaller effects, $+0.44$ on GPT-5-mini). Of the 15 pairwise correctness comparisons, nine survive Holm correction, all in the direction that preservation hurts. These tests treat runs as independent; treating the 24 tasks as the unit of analysis instead, the preservation effect remains significant on GPT-5-mini and Qwen3, is borderline on Gemini, and is not significant on Claude, whose drop comes from four tasks (Appendix B.6). The GPT-5-mini effect is more than 2.5$\times$ the others; it is the only OpenAI model in the panel and the only one forced to `temperature=1` (Section 7).

Finding 2: Shyness moves in the same direction, but is a first-order effect only on GPT-5-mini. Shyness rises under preservation on all five models. On the four non-OpenAI models the effect is small ($|d| \leq 0.067$) and no baseline-vs-preservation shyness contrast survives correction. On GPT-5-mini it doubles from 13 % to 26 % ($p_{\text{Holm}} = 0.046$, $d = -0.129$). Two deletion-vs-preservation shyness contrasts also survive: Gemma 4 ($p_{\text{Holm}} = 0.011$) and GPT-5-mini ($p_{\text{Holm}} < 10^{-13}$). Correctness is the headline; shyness is a smaller mechanism within it on most models.

Finding 3: Wrapping is the baseline failure mode; downstream patching is the preservation-amplified one. Under neutral prompts, 30–37% of correct B1 (wrapping) patches are additive on

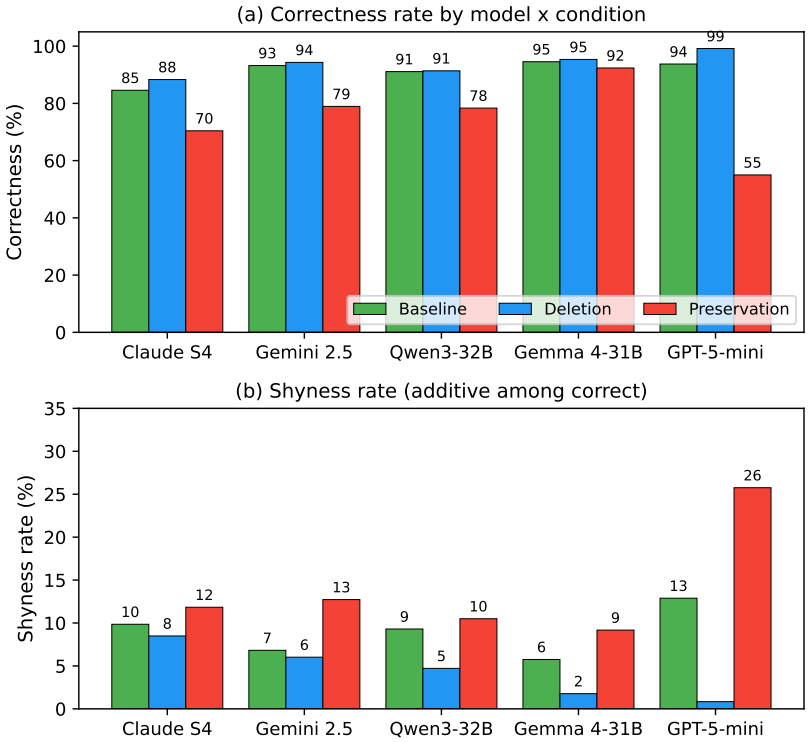

Figure 1: All five models under three conditions (baseline: green; deletion: blue; preservation: red). **(a)** Correctness rate: preservation degrades correctness vs. baseline on all five models, significantly on four of five (Gemma 4 not significant), with GPT-5-mini's drop largest; deletion is neutral-to-positive. **(b)** Shyness rate (additive share among correct patches): preservation raises it on every model, most steeply on GPT-5-mini.

Table 2: Per-bias shyness rate across all models and conditions. Cells show *shy%* (additive among correct); a bracketed $[k]$ indicates k shy-failure meta-test catches (Shy_F). Source: recomputed from the released JSONL files.

	Claude S4			Gemini 2.5			Qwen3-32B			Gemma 4-31B			GPT-5-mini		
Bias	b	d	p	b	d	p	b	d	p	b	d	p	b	d	p
B1	33	37	33	36	31	**45**	33	13	34	37	13	38	30	0	33
B2	0	0	0	0	0	0	0	0	[1]	0	0	0	0	0	[21]
B3	0	0	[10]	0	0	[2]	[2]	26	28[6]	0	0	0	50[10]	7	**53**[13]
B4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	[10]
B5	0	0	0	0	0	6	0	0	0	0	0	0	0	0	**56**[10]
B6	0	0	[9]	0	0	[11]	0	0	0	0	0	0	0	0	[28]
B7	33	27	33	17	17	**38**	14	0	14	10	0	**33**	33	0	34[1]
B8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

b = baseline, d = deletion-encouraging, p = preservation-encouraging. **Bold**: clearest preservation-driven jumps. B1 (wrapping) is roughly constant across conditions; B7 (downstream patching) shows the largest baseline-to-preservation jump.

all five models, and preservation barely changes this (33–45%)—wrapping is the default failure mode of code-generating LLMs in our data. Downstream patching (B7) is what preservation amplifies: it climbs from 17% to 38% on Gemini and from 10% to 33% on Gemma 4 (Claude's is already 33% at baseline; Qwen3's stays flat). On GPT-5-mini, preservation instead amplifies B3 duplication ($50\% \rightarrow 53\%$) and B5 abstraction inflation ($0\% \rightarrow 56\%$), plus many shy-failure meta-test catches (28 under B6) (Table 2; further per-bias analysis in Appendix B.4).

Table 3: Deletion ratio by AI tool (AIDev, 42.5K commits) vs. human baseline (14.8K commits). MW p: one-sided Mann-Whitney (human $>$ AI). d: Cliff's delta, not computed for non-significant comparisons (n/a).

Tool	N	Med. δ	$\delta \geq 0.5$	MW p	d
Codex	21,023	**.167**	18.1%	$<\mathbf{10}^{-300}$	.27
Claude	1,304	**.258**	26.8%	$\mathbf{10}^{-13}$	.13
Cursor	3,225	.345	34.2%	.42	n/a
Copilot	7,928	.399	38.8%	1.0	n/a
Devin	9,053	.395	38.6%	1.0	n/a
Human	14,838	.356	31.5%		

Finding 4: The deletion instruction is neutral-to-positive. Deletion never significantly hurts correctness on any model. It gives the lowest shyness on Gemma 4 (2%, vs. 6% at baseline and 9% under preservation; significant vs. preservation, $p_{\text{Holm}} = 0.010$) and a directional 3-pp correctness gain on Claude. In our setting, deletion is at minimum a safe choice; preservation is not.

5 Observational Evidence in Deployed AI Tools

We next ask whether the pattern is visible in deployed code, comparing the per-commit deletion ratio δ of AI-attributed and human commits from the same repositories. This analysis is consistent with, but cannot causally isolate, the prompt-level effect. From the AIDev dataset Li et al. (2026) (933K AI-attributed PRs, five tools spanning CLI agents, IDE assistants, and an autonomous agent) we obtain **42,533** merged AI commits, plus **14,838** human commits fetched via the GitHub API for AIDev's human PRs: **57,371 commits** with ≥ 5 changed lines (Appendix C).

Overall, human commits delete more than AI commits (Mann-Whitney $p < 10^{-104}$; median δ 0.356 vs. 0.261), but the aggregate hides large per-tool variation (Table 3, Figure 2). **Codex and Claude Code delete significantly less than humans**; Cursor matches humans; Copilot and Devin delete *more*. The two low extremes differ relative to their guidance. Codex's guidance leans toward preservation OpenAI (2025), so its low ratio is in the predicted direction, although the shipped instruction is narrower than the one we test. Claude Code is more striking: its prompt licenses deleting unused code Anthropic (2026), yet it still deletes below the human rate—consistent with base-model shyness that the system prompt does not fully overcome. The tools also differ in interaction paradigm, context exposure, and base model, so we present these patterns as *consistent with* the controlled-study mechanism, not as independent causal evidence. Effect sizes are modest ($d = 0.27$ and 0.13); the tiny p-values come from large N. Task-type labels show AI tools *can* delete when the task demands it (refactoring commits have $\delta \geq 0.5$ in 50.9%), and a small SWE-bench Verified sanity check shows the same directional shift under preservation (Appendix C.3).

6 Toward a Scope-Limited Preservation Instruction

Both shipped instructions are unconditional. A natural *Pareto-improving* alternative would scope preservation to user-facing behavior while permitting internal cleanup ("preserve the existing public API and observable behavior; freely delete internal scaffolding, dead code, and helpers that no live caller references"). In a pre-registered Phase 2 we tested six such candidates (P1–P6: API scoping, behavior preservation, risk tiering, liveness, a hybrid, and test anchoring) against blanket preservation and baseline on Qwen3-32B, Gemma 4-31B, and Llama 4. Each cell ran 36 tasks—the 24 ShyBench tasks plus 12 *benefit-measuring* (BM) tasks whose gold fixes preserve a documented public API and so measure over-deletion—at n=5. The pre-registered gate requires (i) mean correctness recovery $\geq$75 % of the blanket–baseline gap, (ii) BM over-deletion within 1 pp of blanket on every model, and (iii) both in the same direction on every model.

All six candidates recover correctness on average (80 % for P5 to 92 % for P1) and improve on blanket on every model, but **none clears the gate**: on Llama 4 every prompt that licenses internal cleanup shows +7 to +17 pp more BM over-deletion than blanket preservation (Figure 3). At 60 runs per cell these increases are individually small—none is significant after Holm correction across the

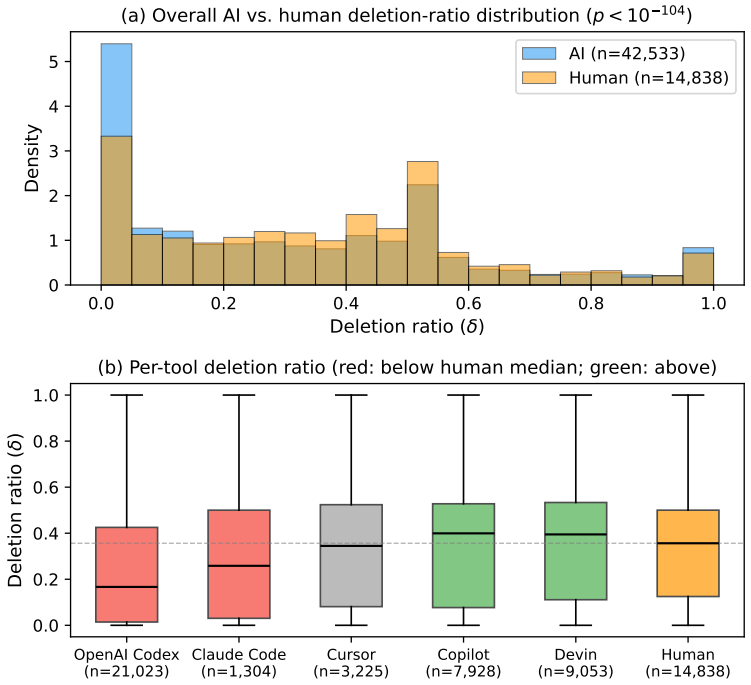

Figure 2: (a) Overall AI (42.5K, blue) vs. human (14.8K, orange) deletion ratio distributions, drawn semi-transparent so their overlap appears olive ($p < 10^{-104}$). (b) Per-tool box plots, colored by position relative to the human median (red: below; gray: at it; green: above; orange: the human baseline): Codex and Claude Code fall below, Cursor sits at the median, and Copilot and Devin above. The two low-deleters' guidance leans in opposite directions—Codex's toward preservation, Claude Code's toward deletion—yet both land below humans (Section 5).

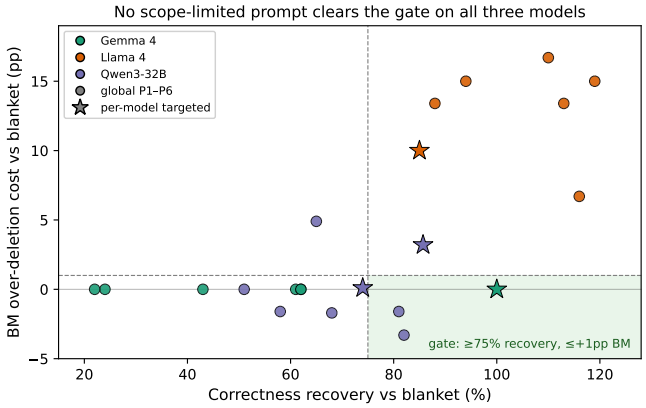

Figure 3: Phase 2: correctness recovery vs. BM over-deletion cost, one point per (candidate, model). Circles: global candidates P1–P6; stars: per-model targeted prompts. Shaded: the pre-registered gate. A prompt qualifies only if all three of its model points fall inside; none do. Llama 4 (orange) shows 7–17 pp more BM over-deletion than under blanket preservation regardless of recovery; no single increase is significant after Holm correction (Appendix D).

six candidates ($p_{\text{Holm}} \geq 0.15$)—so the gate result is "not shown to be safe" rather than "shown to be unsafe." Because a 30-diff sample showed the models fail differently (Llama 4 over-deletes meta-content, Gemma 4 over-preserves scaffolding, Qwen 3 makes logic-ordering errors), we also tried exploratory per-model *targeted* prompts. Gemma 4 fully recovers with no BM cost, but Llama 4

recovers 85 % only at +10 pp BM cost, and Qwen 3 either stops at 74 % recovery with safety intact or reaches 85.7 % at +3.2 pp BM cost. Where blanket preservation imposes a meaningful cost, scope-limited preservation can recover most of it or hold the safety property, but we found none that does both; with this sample size, that is consistent with a trade-off but does not establish one. Details are in Appendix D.

7 Discussion and Limitations

What the controlled study supports, and what it does not. The controlled study supports a clear claim: a short, explicit preservation instruction appended to the request reduces patch correctness on every LLM we tested, by 13–15 pp on three of five models and by 39 pp on GPT-5-mini (the only OpenAI model in our panel), surviving Holm correction across the full 30-test grid when runs are the unit of analysis (at the task level it holds on GPT-5-mini and Qwen3, is borderline on Gemini, and is not significant on Claude; Appendix B.6), with Cliff's d of $+0.13$ to $+0.14$ on three baseline-vs.-preservation contrasts (up to $+0.18$ on the steeper deletion-vs.-preservation contrast for those same models, and up to $+0.44$ on GPT-5-mini). This is the core empirical contribution. It does *not* support the broader claim that preservation prompts cause all observed deletion-shyness behavior in deployed tools, nor does it measure the vendors' verbatim instructions: the deployed tools include far more than one instruction, the shipped wording is narrower than ours, and our 24-task benchmark is synthetic.

What the observational study suggests, with caveats. The two tools with the lowest deletion ratios in 42,533 deployed AI commits are Codex and Claude Code. For Codex, whose guidance leans toward preservation, this direction is what the controlled-study mechanism predicts. Claude Code is the more interesting case: its vendor's prompt licenses deletion, yet it still deletes below the human rate—which we read as base-model shyness surviving a prompt that pushes the other way, rather than as prompt-driven preservation. The effect sizes (Cliff's $d = 0.27$ for Codex, 0.13 for Claude Code) are small to small-medium; the very small p-values come primarily from large N, not large per-commit gaps. We do not control for interaction paradigm, task distribution, model identity, or context exposure across the five tools, so the observational pattern can support only an internal-consistency check, not an independent causal argument.

Why wrapping is the universal baseline failure mode. B1 (wrapping) shyness rates are 30–37% under neutral prompts across *all five* models, barely moving under preservation. This is a stronger finding than what the original Gemini-only analysis suggested. Wrapping appears to be the default "safe" modification move that code-generating LLMs reach for when faced with buggy code: add a guard, an error handler, a fallback. Removing the root cause requires extra confidence that the LLM does not muster by default. B7 (downstream patching) is the bias preservation most activates—on Gemini and Gemma 4 it climbs sharply, rerouting the fix *further away* from the bug, where it is harder to detect and harder to remove (Claude's B7 is already high without the instruction, and Qwen3's is unmoved). On GPT-5-mini, preservation instead amplifies B3 (duplication: creating parallel `_v2` functions) and B5 (abstraction inflation): the model adds layers around the bug rather than rewiring downstream.

The deletion-vs.-preservation question in practice. If we treat the controlled study as causal, then our preservation instruction imposes a measurable cost on every model tested, and our deletion instruction does not. From an engineering-design standpoint, the deletion-instruction default is strictly safer in our setting. We are careful with the word "strictly": our setting is a 24-task benchmark of subtractive fixes. A future test on a benchmark of tasks where the gold fix preserves an existing API is needed before claiming that the deletion instruction is universally safer. We flag this as the most important follow-up experiment.

Limitations. (0) *Prompt fidelity:* our instructions are stronger, more general versions of the vendor guidance, delivered at the end of the user turn rather than in a system prompt; the effect of the verbatim shipped wording, in a system prompt, is untested. (1) *Decoding asymmetry:* GPT-5-mini's 39 pp effect could partly reflect its forced `temperature=1`; the direction and significance do not depend on the magnitude. (2) *Benchmark construction:* gold fixes are subtractive by design, which mechanically favors the deletion instruction; the BM tasks provide the complementary test. (3)

Scope: 24 single-file Python tasks, single-turn prompting, and no test inside the full Codex scaffolding; GPT-5-mini is a mid-tier model, and Phase 2 uses local models only. (4) *Attrition:* counting all dropped runs as failures preserves every correctness effect. (5) *Observational confounds:* tool paradigm, base model, and user population are not controlled. (6) *Correctness includes meta-tests:* a patch that works but keeps code the task asked to remove (e.g., a dead function) fails, so part of the preservation effect reflects compliance with the instruction rather than broken behavior; our saved outputs record only overall pass/fail, so functional-only correctness cannot be separated out without re-running. (7) *Task clustering:* the headline tests treat runs as independent; at the task level the Claude effect is not significant and the Gemini effect is borderline (Appendix B.6). A full threats-to-validity analysis is in Appendix E.

8 Conclusion

The two largest AI coding tools give guidance that leans in opposite directions on removing code. In 3,600 controlled runs across five LLMs, with an explicit, generalized version of each as the only manipulation, the preservation instruction reduces patch correctness on every model—by 13–15 pp on three of five and by 39 pp on GPT-5-mini—an effect that is robust at the task level on GPT-5-mini and Qwen3, while the deletion instruction is neutral-to-positive. In 57,371 deployed commits the lowest deletion ratios belong to Codex and to Claude Code, the latter despite a prompt that licenses deletion. The two kinds of guidance are therefore not interchangeable: on every model and condition we tested, preservation is the worse choice or a tie, and no scope-limited rewording we tested cleared its pre-registered gate. Whether the narrower shipped wording carries the same cost, and how much of the deployed-tool deletion gap it explains, remain open. We release the tasks, the ≈8,600 archived runs, and the pipeline that regenerates every table and figure.

Use of Generative AI

The authors used Claude Code (an AI coding assistant from Anthropic) during the preparation of this work to assist with experiment runner and analyzer implementation, statistical pipeline code, prose editing, and figure generation. The authors verified all code, all numerical results reported in the paper, and all prose, and are responsible for the content of this paper. Note that Claude Code is itself one of the AI coding tools studied in the observational analysis (Section 5); this tool choice was made independently of the experimental design and did not influence the interpretation of results.

References

Gabrielle S. Adams, Benjamin A. Converse, Andrew H. Hales, and Leidy E. Klotz. People systematically overlook subtractive changes. *Nature*, 592:258–261, 2021. doi: 10.1038/s41586-021-03380-y. URL https://doi.org/10.1038/s41586-021-03380-y.

Anthropic. Claude sonnet 4. Technical report, Anthropic, 2025. https://www.anthropic.com/news/claude-4.

Anthropic. Claude code system prompt. Leaked source code, March 31, 2026. Documented in VentureBeat and archived at https://github.com/Piebald-AI/claude-code-system-prompts, 2026. Analysis: https://www.dbreunig.com/2026/04/04/how-claude-code-builds-a-system-prompt.html.

baran312. Cursor AI dead code problem: 30% of my 6,000 lines are trash. Cursor Forum, 2025. https://forum.cursor.com/t/cursor-ai-dead-code-problem-30-of-my-6000-lines-are-trash-need-help-for-permanent-cleanup/96964.

Jan Betley, Christian Warncke, Alicja Sztyber-Betley, Daniel Tan, Niels Bao, Martín Soto, Anna Srivastava, Nathan Labenz, and Owain Evans. Training large language models on narrow tasks can lead to broad misalignment. *Nature*, 2026. doi: 10.1038/s41586-025-09937-5. URL https://doi.org/10.1038/s41586-025-09937-5.

Vanessa Cheung, Maximilian Maier, and Falk Lieder. Large language models show amplified cognitive biases in moral decision-making. *Proceedings of the National Academy of Sciences*, 2025.

doi: 10.1073/pnas.2412015122. URL https://doi.org/10.1073/pnas.2412015122.

Jonathan Cordeiro, Shayan Noei, and Ying Zou. An empirical study on the code refactoring capability of large language models. *arXiv preprint arXiv:2411.02320*, 2024.

DoltHub. Claude code gotchas. DoltHub Blog, 2025. https://www.dolthub.com/blog/2025-06-30-claude-code-gotchas/.

Jessica M. Echterhoff, Yao Liu, Abeer Alessa, Julian McAuley, and Zexue He. Cognitive biases in large language models: A survey and mitigation experiments. In *ACM/SIGAPP Symposium on Applied Computing*, 2025. doi: 10.1145/3672608.3707812. URL https://doi.org/10.1145/3672608.3707812.

Aaron Fanous, Jacob Goldberg, Ank A. Agarwal, Joanna Lin, Anson Zhou, Roxana Daneshjou, and Sanmi Koyejo. SycEval: Evaluating LLM sycophancy. In *AAAI/ACM Conference on AI, Ethics, and Society (AIES)*, 2025. arXiv:2502.08177.

GitClear. Coding on copilot: Data shows AI's downward pressure on code quality. Technical report, GitClear, 2024. https://www.gitclear.com/coding_on_copilot_data_shows_ais_downward_pressure_on_code_quality.

GitClear. AI copilot code quality 2025 research. Technical report, GitClear, 2025. https://www.gitclear.com/ai_assistant_code_quality_2025_research.

Google DeepMind. Gemini 2.5: Our most intelligent AI model. Technical report, Google DeepMind, 2025. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/.

Google DeepMind. Gemma 4 technical report. Technical report, Google DeepMind, April 2026. Open-weight model family (E2B, E4B, 26B MoE, 31B dense); Apache 2.0 license; 256K context for 26B and 31B variants. https://deepmind.google/models/gemma/gemma-4/.

Google DORA Team. 2025 DORA state of AI-assisted software development report. Technical report, Google Cloud, 2025. https://cloud.google.com/resources/content/2025-dora-ai-assisted-software-development-report.

Tiansheng Huang, Sihao Hu, Fatih Ilhan, Selim Furkan Tekin, Zachary Yahn, Yichang Xu, and Ling Liu. Safety tax: Safety alignment makes your large reasoning models less reasonable. *arXiv preprint arXiv:2503.00555*, 2025.

Carlos E. Jimenez et al. SWE-bench: Can language models resolve real-world GitHub issues? In *ICLR*, 2024.

Andrej Karpathy. Notes on coding with AI agents. X (Twitter), 2026. https://x.com/karpathy/status/2035173492447224237.

Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. AIDev: Studying AI coding agents on GitHub. arXiv preprint arXiv:2602.09185, 2026. Dataset: 932,791 agent-authored pull requests from five tools (Codex, Devin, Copilot, Cursor, Claude Code) across 116,211 GitHub repositories; curated 33,596-PR subset with rich metadata.

C. Montes. Lint against the machine: A field guide to catching AI coding agent anti-patterns. Medium, 2026. https://medium.com/@montes.makes/lint-against-the-machine-a-field-guide-to-catching-ai-coding-agent-anti-patterns-3c4ef7baeb9e.

Daniel Ogenrwot and John Businge. How AI coding agents modify code. In *MSR 2026 Mining Challenge*, 2026. arXiv:2601.17581.

OpenAI. Codex system prompt and prompting guide. https://developers.openai.com/cookbook/examples/gpt-5/codex_prompting_guide, 2025. Includes system-level instructions for code preservation behavior.

OX Security. OX report: AI-generated code violates engineering best practices. PR Newswire, 2025. https://www.prnewswire.com/news-releases/ox-report-ai-generated-code-violates-engineering-best-practices-undermining-software-security-at-scale-302592642.html.

Qwen Team, Alibaba Cloud. Qwen3 technical report. Technical report, Alibaba Cloud, 2025. https://qwenlm.github.io/blog/qwen3/.

Amirali Sajadi, Kostadin Damevski, and Preetha Chatterjee. How safe are AI-generated patches? A large-scale study on security risks in LLM and agentic automated program repair on SWE-bench. *arXiv preprint arXiv:2507.02976*, 2025.

Luca Santagata and Cristiano De Nobili. More is more: Addition bias in large language models. *Computers in Human Behavior: Artificial Humans*, 3:100129, 2025. doi: 10.1016/j.chbah.2025.100129. URL https://doi.org/10.1016/j.chbah.2025.100129. arXiv:2409.02569.

Mrinank Sharma et al. Towards understanding sycophancy in language models. *arXiv preprint arXiv:2310.13548*, 2023.

SonarSource. The inevitable rise of poor code quality in AI-accelerated codebases. SonarSource Blog, 2025. https://www.sonarsource.com/blog/the-inevitable-rise-of-poor-code-quality-in-ai-accelerated-codebases/.

Lydia Uhler, Verena Jordan, Jurgen Buder, Markus Huff, and Frank Papenmeier. Influence of solution efficiency and valence of instruction on additive and subtractive solution strategies in humans, GPT-4, and GPT-4o. *Communications Psychology*, 2026. doi: 10.1038/s44271-026-00403-0. URL https://doi.org/10.1038/s44271-026-00403-0. arXiv:2404.16692.

APPENDIX OVERVIEW

- Appendix A: The eight additive failure modes (B1–B8).
- Appendix B: Controlled-study details—benchmark construction, model configuration, statistical analysis, per-bias results, robustness checks, and a task-clustered re-analysis.
- Appendix C: Observational-study details—data construction, task types, the SWE-bench sanity check, and practitioner reports.
- Appendix D: Implications for system-prompt design, and Phase 2 details—global scope-limited candidates and per-model targeted prompts.
- Appendix E: Threats to validity.

A THE EIGHT ADDITIVE FAILURE MODES

To diagnose *how* preservation-style instructions degrade correctness, we need a vocabulary for the additive patterns that replace a clean subtractive fix when the model tries to avoid deletion. We organize these into eight failure modes—the bias categories B1–B8, each a way the additive ("shy") tendency shows up in code—spanning two tiers: *surface failures* (directly visible in patches) and *structural failures* (disguised as legitimate engineering practice). We treat this taxonomy as an analytical instrument rather than a primary contribution; the labels also align with practitioner reports of AI code anti-patterns Karpathy (2026); Montes (2026); SonarSource (2025); OX Security (2025).

A.1 SURFACE BIASES

A.1.1 B1: WRAPPING INSTEAD OF FIXING

LLMs wrap buggy code in `try/catch` blocks, error handlers, or compatibility shims rather than removing the root cause. The bug persists underneath a layer of defensive logic. For example, a function with an off-by-one error gets wrapped in bounds checking instead of having the index corrected.

A.1.2 B2: COMMENTING OUT INSTEAD OF DELETING

When code should be removed, LLMs comment it out or add `// removed` markers instead of deleting it. Version control already provides a complete history; commented-out code serves no purpose that `git log` does not already serve better.

A.1.3 B3: DUPLICATION INSTEAD OF REPLACEMENT

Rather than replacing a broken function, LLMs introduce a new variant alongside the original—`processData` and `processDataV2`. Both remain in the codebase, confusing future developers about which is canonical.

A.1.4 B4: HOARDING DEAD CODE

LLMs preserve unused imports, unreachable branches, and dead feature flags "just in case," degrading readability and inflating the test surface.

A.2 STRUCTURAL BIASES

The following four biases are more insidious because they often resemble legitimate engineering patterns.

A.2.1 B5: ABSTRACTION INFLATION

Rather than simplifying messy code, LLMs add a new abstraction layer *on top*—an adapter class, a facade—hiding complexity without removing it. Karpathy noted agents "implement a bloated construction over 1000 lines when 100 would do" Karpathy (2026).

A.2.2 B6: CONFIGURATION OVER DELETION

Instead of removing obsolete behavior, LLMs make it toggleable via a feature flag or `USE_LEGACY_MODE` parameter. The dead code path lives on as a configurable option that no one will ever enable. This is particularly harmful because it suggests to future maintainers that the old behavior is *intentionally preserved*.

A.2.3 B7: DOWNSTREAM PATCHING

LLMs add correction logic *downstream* rather than fixing the *upstream* root cause. The bug remains; its symptoms are masked three function calls later. This is distinct from B1 because the wrapping happens far from the bug.

A.2.4 B8: DEPRECATION THEATER

LLMs add `@deprecated` annotations or `// TODO: remove` markers as a *substitute* for actual deletion—a social gesture of acknowledgment without action. These markers accumulate: a codebase can accrue dozens of `// TODO: remove this` comments, each representing a deletion deferred indefinitely.

B CONTROLLED-STUDY DETAILS

B.1 BENCHMARK CONSTRUCTION

ShyBench contains 24 Python tasks (three per bias category B1–B8). Each task consists of:

1. A **buggy code snippet** (50–200 lines) with a known defect.
2. A **gold subtractive fix**: the minimal correct patch, requiring deletion or simplification.
3. A **plausible additive fix**: a functionally correct but bloated alternative that wraps, duplicates, or extends.
4. A **test suite** with *functional tests* (verify correctness) and *meta-tests* (detect shy patterns—e.g., checking that `try/except` was not added for B1, that no `_v2` function was created for B3, that dead code was actually removed for B2/B4/B6/B8).

Meta-tests strip docstrings and comments via regex before pattern matching, to avoid false positives from task descriptions in the source.

B.2 MODEL CONFIGURATION

We test five LLMs spanning five model families and four providers: Claude Sonnet 4 Anthropic (2025) (Anthropic, via API), Gemini 2.5 Pro Google DeepMind (2025) (Google, via API), GPT-5-mini (OpenAI, via API), and two open-weight models, Qwen3-32B Qwen Team, Alibaba Cloud (2025) (Alibaba) and Gemma 4-31B Google DeepMind (2026) (Google DeepMind), run locally via Ollama at Q4_K_M quantization. Each model is tested with n=10 runs per task-condition, yielding 720 runs per model and **3,600 total**. Claude Sonnet 4 and GPT-5-mini are the model families behind Claude Code and Codex respectively—the two production tools whose deployed deletion behavior we analyze in Section 5—directly linking the observational and controlled studies. The GPT-5-mini API requires `temperature=1` and exposes a `reasoning_effort` parameter that consumes output tokens for hidden reasoning before code generation; we set `reasoning_effort='low'` to preserve the visible completion-token budget for the patch. The other four models run at `temperature=0.7` per the existing Phase 1 configuration; we treat this asymmetry as a threat to internal comparability and discuss it in Section E.

B.3 STATISTICAL ANALYSIS

Each outcome (correctness, shyness) is binary per run, so a pairwise comparison between two conditions is a 2×2 contingency table, which we test with **Fisher's exact test** (two-sided)—an exact test that makes no large-sample approximation and is appropriate at our per-cell n=10. For effect size

we report **Cliff's** d, a non-parametric measure equal to the probability that a run in one condition outranks a run in the other minus the reverse probability ($d \in [-1, 1]$, 0 meaning no difference); on binary outcomes it reduces to the difference in proportions. For a comparison "A vs. B" (e.g., baseline vs. preservation), d is computed as A minus B, so $d > 0$ means A has the higher rate: a positive correctness d means A is more often correct, and a negative shyness d means A is less often shy. The same convention applies to the observational study, where A is the human baseline. We write d throughout; δ is reserved for the deletion ratio. We report d for *every* comparison, significant or not, so magnitude can be judged separately from significance. Because we run a grid of many tests (5 models $\times$ 2 outcomes $\times$ 3 condition pairs, 30 in total), raw p-values would overstate significance, so we apply **Holm correction**: a sequentially-rejective procedure that controls the family-wise error rate (the probability of *any* false positive across the grid) and is uniformly more powerful than Bonferroni under the same minimal assumptions. We write p_{Holm} for the resulting adjusted p-values and call a comparison significant when $p_{\text{Holm}} < 0.05$. The more lenient Benjamini–Hochberg FDR correction is also computed and yields the same qualitative set of significant cells. The released analysis script regenerates every p-value and effect size from the raw JSONL.

Effect sizes omitted from the main text. For the two deletion-vs.-preservation shyness contrasts that survive Holm correction (Finding 2), Cliff's $d = -0.074$ on Gemma 4 ($p_{\text{Holm}} = 0.011$) and $d = -0.249$ on GPT-5-mini ($p_{\text{Holm}} < 10^{-13}$). For Gemma 4's shyness under the deletion instruction (Finding 4), the deletion-vs.-preservation reduction is significant ($p_{\text{Holm}} = 0.010$), but the deletion-vs.-baseline change is not ($p_{\text{Holm}} = 0.67$).

B.4 Per-Bias Results

Table 2 shows the per-bias shyness rate across every (model, condition) cell; Figure 4 visualizes the dominant per-bias jumps for Gemini 2.5 Pro. Two patterns hold across all five models:

Wrapping (B1) is a baseline behavior. Under neutral prompts, 30–37% of B1 correct patches are additive across all five models; the preservation instruction barely changes this (33–45%). This is the universal default failure mode of code-generating LLMs in our data, present even without preservation guidance.

Downstream patching (B7) is preservation-amplified on two models. Under neutral prompts, B7 shyness is 10–17% on Gemini, Qwen3, and Gemma 4 (and already 33% on Claude S4); under preservation it climbs sharply on Gemini (17% $\rightarrow$ 38%) and Gemma 4 (10% $\rightarrow$ 33%), while Qwen3 stays flat (14%) and Claude remains at its elevated baseline. GPT-5-mini already has an elevated baseline B7 rate (33%) that stays roughly flat under preservation, but its preservation-amplified failures appear under other categories (B3 duplication: 50 % $\rightarrow$ 53 %, B5 abstraction inflation: 0 % $\rightarrow$ 56 %) plus a heavy load of shy-failure meta-test catches (28 under B6, 21 under B2, 13 under B3). Where preservation specifically degrades model behavior, the dominant mode is displacing the fix away from the root cause on the non-OpenAI panel and duplicating/inflating-around the bug on GPT-5-mini.

The other six bias categories show low or zero shyness across most models, with one preservation-specific failure mode that does not manifest as shy-correct but as shy-failure: configuration over deletion (B6) under preservation produces 9–11 shy meta-test failures for Claude and Gemini (the model preserves the dead code by making it toggleable, but fails the functional tests).

B.5 Robustness Checks

Sample-loss sensitivity. The preservation condition has the highest dropout for Gemini (31 of 240 runs returned errors or empty content). Treating all dropped runs as incorrect rather than excluding them yields a baseline-vs-preservation correctness gap of -23 pp for Gemini (versus -14 pp in Table 1), -15 pp for Claude and Qwen3 (unchanged or larger), and -3 pp for Gemma 4 (unchanged). The correctness effect is preserved or strengthened on every model under the worst-case assumption, ruling out asymmetric attrition as the source of the finding.

Threshold robustness. Reclassifying patches under deletion-ratio thresholds $\delta \in \{0.2, 0.3, 0.4, 0.5\}$ yields identical shyness rates in every (model, condition) cell, because ad-

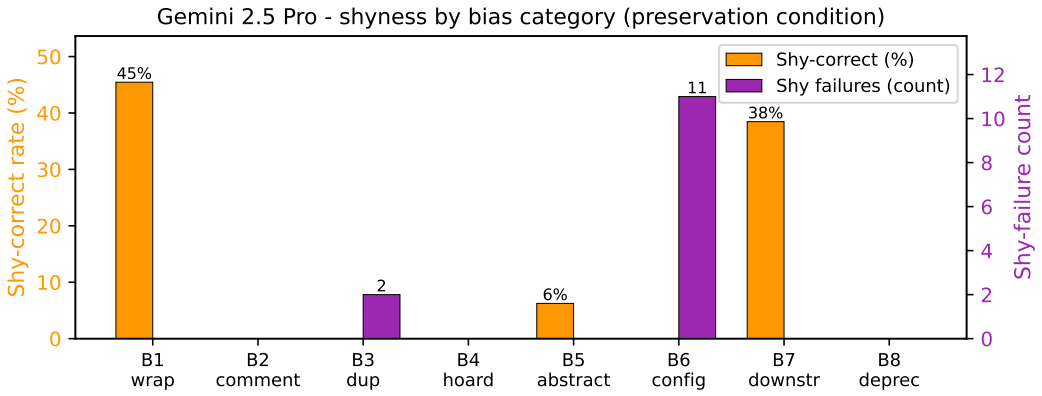

Figure 4: Per-bias shyness under preservation for Gemini 2.5 Pro (illustrative). Two measures per bias category on separate axes: shy-correct rate (orange, left axis—% of correct patches that are additive) and shy-failure count (purple, right axis—incorrect low-deletion patches caught by meta-tests). The shy-correct rate concentrates in B1 (wrapping, 45%) and B7 (downstream patching, 38%); the cross-model pattern is shown in Table 2.

ditive patches have $\beta > 0$ and are therefore excluded from *subtractive* regardless of δ. The headline numbers are not sensitive to the threshold parameter.

B.6 Task-Clustered Analysis

The Fisher tests in Section 4 treat the ≈240 runs in each (model, condition) cell as independent. They are 10 runs on each of 24 tasks, so runs on the same task are correlated and the effective sample size is closer to the number of tasks. We therefore re-test every correctness contrast with the task as the unit of analysis, in three ways: a GEE logistic regression with an exchangeable working correlation, clustered by task; a cluster bootstrap over tasks (10,000 resamples of the 24 tasks) for a 95% interval on the pooled difference; and a paired Wilcoxon signed-rank test on per-task correctness rates. p-values are Holm-corrected across the 15 correctness contrasts, separately for each method. Table 4 reports the results.

The preservation effect remains significant at the task level on GPT-5-mini (15 of 24 tasks worse under preservation, none better) and Qwen3-32B (12 worse, none better), and is borderline on Gemini 2.5 Pro (significant under GEE, not under Wilcoxon; 11 worse, 1 better). On Claude Sonnet 4 it is not significant at the task level: the 14 pp pooled drop comes from four tasks (B2-002, B3-001, B5-002, B6-001) that fall from 100 % correct to 0–10 %, while one task improves. The deletion instruction does not significantly change correctness on any model at the task level, consistent with Finding 4. The analysis script is `experiments/phase1_clustered.py` in the artifact.

Table 4: Task-clustered re-analysis of Phase 1 correctness. Diff: pooled correctness rate of A minus B (pp), over the valid runs used in Table 1. CI: 95% cluster-bootstrap interval over tasks. GEE and Wilcoxon: Holm-corrected p-values across the 15 contrasts (bold: < 0.05). Tasks: number of tasks where A is higher / B is higher / tied.

Model	A vs B	Diff	95% CI	GEE p_{Holm}	Wilcoxon p_{Holm}	Tasks
Claude S4	base vs pres	+14.2	[+0.0, +30.8]	0.56	0.59	4 / 1 / 19
Claude S4	del vs pres	+17.9	[+0.0, +36.7]	0.48	0.42	6 / 2 / 16
Claude S4	base vs del	−3.7	[−15.0, +4.2]	1	1	2 / 2 / 20
Gemini 2.5	base vs pres	+14.3	[+3.9, +27.2]	**0.048**	0.072	11 / 1 / 12
Gemini 2.5	del vs pres	+15.4	[+4.5, +29.1]	**0.031**	0.12	10 / 3 / 11
Gemini 2.5	base vs del	−1.1	[−5.6, +2.6]	1	1	4 / 3 / 17
Qwen3-32B	base vs pres	+12.7	[+4.6, +23.4]	**0.037**	**0.028**	12 / 0 / 12
Qwen3-32B	del vs pres	+13.0	[+3.9, +24.4]	**0.039**	0.086	11 / 3 / 10
Qwen3-32B	base vs del	−0.3	[−4.2, +3.4]	1	1	5 / 5 / 14
Gemma 4-31B	base vs pres	+2.2	[−0.8, +7.1]	1	1	3 / 1 / 20
Gemma 4-31B	del vs pres	+3.0	[−1.4, +8.9]	1	1	3 / 1 / 20
Gemma 4-31B	base vs del	−0.8	[−3.7, +1.3]	1	1	1 / 1 / 22
GPT-5-mini	base vs pres	+38.7	[+22.9, +55.0]	7×10^{-5}	9×10^{-3}	15 / 0 / 9
GPT-5-mini	del vs pres	+44.2	[+28.3, +60.8]	1×10^{-7}	6×10^{-3}	16 / 1 / 7
GPT-5-mini	base vs del	−5.4	[−12.5, +0.0]	0.20	0.85	2 / 5 / 17

C Observational-Study Details

C.1 Data Construction

We leverage the AIDev dataset Li et al. (2026), which contains 933K AI-attributed PRs across five tools from 116K repositories. The five tools span distinct interaction paradigms: *CLI agents* that operate in a conversational loop with the developer (OpenAI Codex CLI, Claude Code); *IDE-integrated assistants* that provide inline suggestions and edits (GitHub Copilot, Cursor); and *autonomous agents* that independently plan, execute, and submit complete PRs with minimal human intervention (Devin by Cognition Labs). This diversity is important: if the deletion-ratio gap varies by interaction paradigm, the five tools should reveal it.

After filtering to merged PRs and aggregating file-level statistics to the commit level, we obtain **42,533 AI commits** with per-commit additions/deletions. The dataset includes task-type labels (fix, feature, refactor, docs, test, chore) for 33.6K PRs.

For the human baseline, the AIDev dataset provides 5,081 merged human PRs with repository metadata but without commit-level statistics. We fetched these via the GitHub REST API, obtaining **14,838 human commits** with additions/deletions—yielding a combined dataset of **57,371 commits**. Only commits with ≥ 5 total lines changed were included.

C.2 Task-Type Analysis

The AIDev task-type labels (33.6K PRs) show that refactoring commits have the highest AI deletion rates ($\delta \geq 0.5$ in 50.9%), while test-writing has the lowest (16.9%). Bug fixes fall in between (32.5%). AI tools *can* delete when the task demands it.

C.3 Sanity Check on Real-World Bug Fixes

To check whether the controlled-experiment effect appears in realistic repair tasks, we extracted 8 tasks from SWE-bench Verified Jimenez et al. (2024) where the human gold patch is net-negative (more lines deleted than added), drawn from Django, astropy, pytest, sympy, and matplotlib. We ran Gemini 2.5 Pro on each task (n=3) under baseline and preservation conditions. Under baseline, Gemini matched the human gold deletion ratio on the four tasks where it produced valid responses (mean $\delta = 0.90$ vs. gold 0.91); under preservation, mean δ dropped to 0.83 and one task (SW-008,

matplotlib annotation) shifted from net-deleting to net-adding. We label this a directional sanity check rather than a validation: it is a single model on four parseable tasks, and we discuss the limited generalization in Section 7. A larger-scale SWE-bench validation across the full model panel is future work.

C.4 Practitioner Reports

Developer discourse provides corroboration through reports of specific anti-patterns: Cursor hiding dead UI with CSS `display: none` instead of deleting it (17–30% of code in one audit) baran312 (2025); Claude Code creating parallel `New`-prefixed functions instead of replacing the original DoltHub (2025); Karpathy reporting that agents "implement a bloated construction over 1000 lines when 100 would do" Karpathy (2026); Montes auditing a 130K-line codebase and finding the same rate-limiter implemented four times by the same agent Montes (2026); SonarSource measuring an 8-fold increase in duplicated code blocks from 2020 to 2024 SonarSource (2025); OX Security reporting refactoring avoidance in 80–90% of AI code across 300+ repositories OX Security (2025). These reports are qualitative and source-selected, so we treat them as motivating context rather than evidence. They consistently note that models *can* simplify when explicitly asked—a behavioral default, not a capability limitation.

D System-Prompt Implications and Phase 2 Details

D.1 Implications of the Phase 1 Results

Preservation instructions impose a measurable cost. Our preservation instruction, modeled on Codex's guidance OpenAI (2025), reduces patch correctness by 13–15 pp on three of five models (Claude, Gemini, Qwen3) and by 39 pp on GPT-5-mini—the only OpenAI model in the panel—surviving Holm correction at $\alpha = 0.05$ across the 30-test grid. The effect is consistent in direction across every model in the panel. We did not test the specific Codex prompt against the specific tool's full deployment scaffolding, so we cannot say what fraction of the Codex-vs-Claude-Code observational gap (Section 5) is attributable to this kind of instruction. But the mechanism is plausible at the right magnitude.

Deletion instructions are neutral-to-helpful. Our deletion instruction, modeled on Claude Code's guidance Anthropic (2026), never harms correctness in our data, and yields the lowest shyness for Gemma 4 (significantly below the preservation condition). Whatever risk preservation instructions are presumably guarding against—reckless removal of user-facing behavior—does not visibly manifest as a correctness harm from the deletion instruction in our setting. We caveat that our 24-task benchmark is constructed so the gold fix is subtractive; a benchmark constructed so the gold fix preserves an existing API would be the complementary test of the deletion instruction, and that is a clear direction for future work.

D.2 Global Scope-Limited Candidates

Toward a scope-limited preservation instruction. The two instructions we tested are unconditional. A plausible *Pareto-improving* instruction—one that recovers correctness without worsening any other objective we measure, here baseline-modification safety—would scope preservation to user-facing behavior while permitting internal cleanup: "preserve the existing public API and observable behavior; freely delete internal scaffolding, dead code, and helpers that no live caller references." We tested six concrete variants on this theme as Phase 2 of the present work; we report the result below.

Phase 2 follow-up: global scope-limited candidates. We compared six candidate scope-limited prompts (P1–P6, designed along axes of API scoping, behavior preservation, risk tiering, liveness, a hybrid of the above, and test anchoring) against the blanket-preservation and baseline controls on three locally-hosted models: Qwen3-32B and Gemma 4-31B from the Phase 1 panel, plus Llama 4 (added as a third local model for this follow-up). By *blanket preservation* we mean the original, unscoped preservation instruction tested in Phase 1—the one that forbids deletion unconditionally—as opposed to the scope-limited variants that permit internal cleanup. Each model $\times$ condition cell

ran 36 tasks (the 24 ShyBench items plus 12 *benefit-measuring* tasks BM-001–BM-012, whose gold fixes preserve a documented public API and thereby measure over-deletion; the BM-task count was expanded from 6 in the pre-registration to 12 before pilot completion to broaden the safety measurement) at n=5 runs. The pre-registered pilot-to-production gate has three criteria: (i) the candidate's *mean* correctness recovery across the panel is $\geq 75\,\%$ of the blanket–baseline correctness gap; (ii) the candidate's BM over-deletion rate is within 1 pp of blanket on every model; and (iii) (i) and (ii) hold in the same direction on every panel model. All six candidates recover correctness on average—mean recovery ranges from 80 % (P5 hybrid) to 92 % (P1 API-scoped)—and every candidate improves correctness over blanket on every model in the panel, so the correctness side of the gate is cleared. The blocker is criterion (ii): on Llama 4, every prompt that licenses internal cleanup also licenses a +7 to +17 pp BM over-deletion regression (P2 best at $8.3\,\% \rightarrow 15.0\,\%$, P6 worst at $\rightarrow 25.0\,\%$); P6 also regresses BM safety on Qwen3-32B ($5.1\,\% \rightarrow 10.0\,\%$). 0 of 6 candidates clear the gate (Figure 3).

Statistical power of the BM criterion. Each Llama 4 BM cell has 60 runs (12 tasks, n=5), so one run is 1.7 pp, and the gate's 1 pp tolerance is finer than the data can resolve. Against blanket preservation (5/60 over-deleting), the six candidates have 9–15/60 over-deleting runs; two-sided Fisher tests give $p = 0.026$–0.39 unadjusted and $p_{\text{Holm}} \geq 0.15$ after correction across the six. Llama 4's no-instruction baseline is itself 18.3 % (11/60): blanket preservation *suppresses* over-deletion below baseline, and most candidates return to roughly the baseline level. The gate outcome is therefore best read as "no candidate is shown to be as safe as blanket preservation on Llama 4," not as evidence that the candidates are unsafe. The gate criteria are reported as pre-registered; a follow-up would need roughly an order of magnitude more BM runs to resolve a difference of a few points. The pre-registered Phase 2 gate therefore returns a negative result for a single *global* scope-limited instruction, with the failure mechanism being BM safety on Llama 4 rather than insufficient correctness recovery.

D.3 Per-Model Targeted Prompts

Phase 2 follow-up: per-model targeted prompts. That the global candidates fail only on Llama 4 BM safety motivates relaxing the constraint that one prompt serve all three models. A 30-diff diagnostic sample of failing patches indicates that the three models fail in mechanistically different ways. Llama 4 over-deletes meta-content (docstrings, comments, helpers); Gemma 4 over-preserves scaffolding (leaves orphaned dict keys and references after deleting their referent); Qwen 3 shows no shape bias but commits logic-ordering errors. Drawing one prompt suffix from each failure-mode profile, we evaluated an exploratory per-model *targeted* condition—a separate prompt tailored to each model's failure mode, rather than one shared prompt across all three. Gemma 4—whose blanket–baseline correctness gap is only ≈4 pp to begin with—fully recovers under its targeted prompt with no BM cost ($0\,\% \rightarrow 0\,\%$). The two harder models reproduce the recovery–safety trade-off observed on Llama 4 in the global pilot: Llama 4 under its targeted prompt recovers 85 % of its 27 pp gap but regresses BM by +10 pp ($8.3\,\% \rightarrow 18.3\,\%$); Qwen 3 under its targeted prompt stops at 74 % recovery—a single percentage point short of the 75 % mark a per-model gate would impose—with BM safety preserved ($5.1\,\% \rightarrow 5.2\,\%$), and an exploratory variant that adds a mental-trace verification clause clears 75 % recovery (85.7 %) only by also regressing BM by +3.2 pp ($5.1\,\% \rightarrow 8.3\,\%$). The cross-panel pattern is consistent: for the two models where blanket preservation imposes a meaningful correctness cost, scope-limited preservation can recover most of that cost or hold the safety property, but not both simultaneously. This is consistent with a trade-off on scope-limited preservation in our setting, though the individual BM increases are not significant at this sample size. We have not found a single instruction—global P1–P6 or per-model targeted—that clears the pre-registered gate. All JSONL outputs, prompts, and the analyzer are released with the artifact.

E Threats to Validity

We organize threats under the standard SE-evaluation categories. Many of these threats are addressed in robustness checks (Section B.5); we cross-reference rather than repeat the mitigations here.

Internal validity. Four internal threats deserve attention. First, *asymmetric dropout*: the preservation condition has the highest error rate (31 of 240 runs for Gemini; GPT-5-mini has zero dropout). The sample-loss sensitivity check (Section B.5) treats every dropped run as incorrect; under that worst-case assumption the baseline-vs.-preservation correctness gap is preserved or strengthened on every model, ruling out attrition as the source of the effect. Second, *decoding-parameter asymmetry across the panel*: GPT-5-mini's API requires `temperature=1` and exposes a `reasoning_effort` budget that consumes output tokens; we ran it with `reasoning_effort='low'` to keep the visible budget intact. The other four models ran at `temperature=0.7` with no separate reasoning budget. GPT-5-mini's much larger preservation effect (39 pp vs. 13–15 pp on the other three significantly-affected models) could in principle reflect this decoding asymmetry rather than something model-specific; we cannot disentangle the two with the present data. The direction and the basic Holm-significant result do not depend on the magnitude, so we treat the 39 pp number as conservative for the within-panel comparison and as an interesting open question for follow-up. Third, *benchmark-construction bias*: the 24 ShyBench tasks were constructed by the authors such that the gold fix is subtractive. By that construction, an instruction encouraging deletion is mechanically advantaged on this benchmark. The Phase 2 benefit-measuring tasks (BM-001–BM-012) provide the complementary measurement, where gold fixes preserve a documented public API; the Phase 2 results show that deletion-licensing prompts *do* incur over-deletion costs on these tasks (notably Llama 4 BM over-deletion under all six scope-limited variants), so the benchmark-construction direction does not by itself dictate the overall Phase 2 verdict. Fourth, *taxonomy labeling*: every (task, bias) label was assigned by the paper authors during benchmark construction. The B1–B8 taxonomy is presented as an analytical instrument rather than a primary contribution; the released artifact includes an LLM secondary-coder pipeline with Cohen's κ machinery for independent inter-rater reproduction.

External validity. The controlled study uses 24 single-file Python tasks of 50–200 lines. Multi-file refactoring, other languages (TypeScript, Go, Rust, Java, C++), longer files, and integration-test fixes are not covered. The Phase 1 panel is five models from five families and four providers, including one OpenAI model (GPT-5-mini); GPT-5-mini is OpenAI's mid-tier model rather than the flagship GPT-5, and an OpenAI flagship or reasoning-series model could plausibly behave differently. Our two instructions are not the vendors' verbatim text: they are explicit, general versions (quoted in Section 3) appended to the user turn, whereas the shipped guidance is narrower (Codex's protects a user's uncommitted edits in a dirty worktree; Claude Code's permits deletion only when the model is certain code is unused) and sits in a system prompt. We also did not test the Codex prompt inside the deployed Codex tool, which wraps the instruction in a much larger scaffolding that may strengthen, weaken, or change its effect. The Phase 2 panel is three locally-hosted models—Qwen3-32B and Gemma 4-31B from the Phase 1 panel, plus Llama 4—run locally to keep inference cheap; the two commercial-API models and GPT-5-mini are not included. Phase 2's conclusion about scope-limited prompts is therefore a claim about local models, not the full Phase 1 panel. We test single-turn prompting; production tools use multi-turn agentic loops, and whether such loops compound or attenuate the single-shot effect is open. The observational study cannot isolate prompt content from confounding variables across the five tools (interaction paradigm, base model, context exposure, user population, task distribution); we present it as consistent-with, not as independent causal evidence. AIDev's commit-message-based attribution can miss inline-suggestion contributions from tools like Copilot, and the human baseline was fetched separately via the GitHub API from the same repository pool but with possible composition mismatch.

Construct validity. Because correctness requires passing the meta-tests as well as the functional tests, a patch that follows the preservation instruction by keeping dead code (for example, a meta-test asserting that an obsolete function is no longer defined) is scored incorrect even if it behaves correctly. Part of the measured preservation effect is therefore instruction compliance rather than functional breakage. The saved run records store only overall pass/fail, not the generated code or the failing test, so separating the two requires re-running with outputs saved. Patch correctness is operationalized as passing a task-specific test suite that combines functional tests with meta-tests for shy patterns. This captures correctness against an engineering specification but does not capture stylistic, maintainability, or review-friendliness quality, which are arguably also part of "preservation harm." The meta-tests target named patterns (no `_v2` function for B3, dead code actually removed for B2/B4/B6/B8) and may miss novel evasions of those patterns by the model. Patch classification (additive vs. subtractive) uses $\beta = \text{lines added} - \text{lines deleted}$; a complex one-line replacement can

be classified as additive when it is functionally a refactor, though the threshold-robustness check (Section B.5) shows the headline shyness numbers are not sensitive to the deletion-ratio parameter. The eight failure modes (B1–B8) are a structured grouping of observed patterns rather than a saturating taxonomy; additive failures outside B1–B8 still register correctly in the headline correctness and shyness metrics but would not be diagnosed by the mechanism analysis.

Conclusion validity. The 30-pairwise-test grid is corrected by Holm; Benjamini–Hochberg FDR yields the same qualitative set of significant cells (Section B.5). Effect sizes are reported for every comparison, including non-significant ones, so the reader can judge magnitude separately from significance. Cliff's d of 0.13–0.14 on baseline-vs.-preservation contrasts is small by standard conventions; the substantive scale of the effect is the 13–15 pp absolute correctness gap. The very small p-values in the observational study ($p_{\text{Codex}} < 10^{-300}$) come primarily from large per-tool N rather than per-commit gap; effect sizes (Cliff's $d = 0.13$–0.27) are the more interpretable magnitude there. Phase 2 results are reported against the actual pre-registered three-criterion gate documented in the artifact's design document, committed before the pilot run.