

Does Loop-Back Authority Pay Only When It Can Verify?

A 2×2 Paired Experiment on Opinion- and Verifier-Triggered Revision
in a Production Code-Generation Agent Team

Shinji Yumura
AQUA LLC, Tokyo, Japan
official@aquallc.jp

October 2026

Abstract

Agachan et al. (2026) found that a manager agent with loop-back authority made multi-agent reports slightly worse, and conjectured—without testing it—that such authority “earns its cost when it can verify something and becomes a liability when it can only opine.” We test this conjecture directly in the five-agent team that builds React applications in a publicly deployed assistant, by crossing two independently switchable revision triggers in a 2×2 paired design: an *opinion* trigger (an LLM reviewer that may send files back on its own judgment, up to two rounds) and a *verifier* trigger (the application is executed in a browser preview and runtime errors are sent back, up to two rounds). All four arms ran concurrently on 24 tasks × 2 runs (192 generations) and were judged blind by two LLM judges on five 1–5 dimensions (Krippendorff’s $\alpha = 0.93$ on the overall score) and by a 0/1/2 pass judge. No contrast reached significance for quality: opinion authority changed the overall score by +0.08 (95% CI −0.09 to +0.25) and the pass rate by +4.2 points; verifier authority changed them by +0.06 and +6.2 points. Opinion authority did, however, multiply cost by 3.6×, add 29 s per task, and grow the delivered code by 86 lines (+27%). As in the original study, first drafts were indistinguishable from the flat arm and revision inflated them: lines +21% and defensive constructs per line +42%, a code analogue of the original paper’s +53% hedging. Only 10% of the reviewer’s first-round critiques concerned runtime-checkable defects; 90% were stylistic, aesthetic, or requests for unrequested extensions ($\kappa = 0.91$ between two classifiers). The verifier fired in only 3 of 48 runs, all of which then ran and passed, at negligible cost. A blinded human comparison of 24 A–B pairs, in which the rater used both applications, also found no reliable preference (11 vs. 8, 5 ties; $p = 0.31$). A full replication with a stronger model (`gemini-3.8-flash`, another 192 generations) reproduced the null effect on quality, the cost penalty (3.9×) and the draft inflation (defensive density +52%), with the combined arm tending to perform best (pass 93.8%, not significant). Contrary to the original result we find no quality penalty from opinion authority, but we confirm its mechanism—revision adds material without adding verified value—and show that its main price is paid in money, latency, and code size rather than in judged quality.

1 Introduction

Many LLM agent systems place a manager or reviewer agent above workers and let it reject their output and request revisions. The intuition is borrowed from human organizations: review should catch problems. Whether an LLM reviewer actually improves the work is less clear. Self-correction without external feedback is known to be unreliable [7], whereas correction grounded in tools or execution can help [3, 4, 13].

Agachan, van Duijn and Zohrehvand [1] recently ran a paired experiment on five-role LLM teams writing business-intelligence reports for 43 products. Teams whose manager could force revisions produced reports with lower Utility ($d = 0.42$) and Writing Clarity ($d = 0.34$) than flat teams, with 53% more hedging words; each revision loop cost about 0.14 points of clarity. From the observation that both forms reached ceiling on the one verifiable component (specification accuracy), they conjectured that loop-back authority helps only when the manager can verify something. They did not test this: the trigger of revision was left to the manager, and their first proposed follow-up is to “manipulate the revision trigger (verifier-fired vs. manager-judged loops).”

This paper runs that manipulation. Our setting is a production code-generation team in which two kinds of revision triggers already exist and can be switched independently: an LLM reviewer that judges the code by reading it (*opinion*), and an execution check that runs the generated application and returns runtime errors (*verifier*). Code generation is a natural testbed because a large part of its quality is mechanically checkable. We ask:

1. Does opinion-triggered loop-back authority lower quality in code generation, as it did for reports?
2. Does verifier-triggered loop-back authority raise quality, and do the two interact?
3. Beyond judged quality, what does each kind of authority change—cost, latency, and the content of the code?

Our contributions are: (i) a 2×2 paired experiment that separates opinion- from verifier-triggered revision under equal loop budgets, run concurrently on identical tasks; (ii) a draft-level analysis showing that revision inflates code in a way that mirrors hedging in prose; (iii) a content analysis of what an opinion-only reviewer actually asks for; and (iv) evidence that the cost of unverifiable authority appears in money, latency and code size rather than in judged quality.

2 Related Work

Hierarchy and review in agent teams. Role-based multi-agent frameworks such as MetaGPT [6] and ChatDev [12] include reviewer or tester roles. Recent work questions the value of multi-agent structure itself: a single agent can reproduce many workflows [14], and compute-matched comparisons often favor single-agent baselines [8]. Cemri et al. [2] catalogue failure modes of multi-agent systems; incorrect verification is among the most common. In earlier reports on the same team studied here, we found that most annotated “verification failures” originated in a deterministic check that overwrote the reviewer’s scores rather than in the reviewer model itself [15].

Self-correction. Iterative refinement with self-generated feedback can improve outputs on some tasks [10], but without external signals LLMs often fail to correct, or degrade, their reasoning [7]. Feedback grounded in execution or tools is consistently more useful [3, 4, 13]. Our opinion and verifier triggers map onto this distinction at the level of team authority.

LLM judges. We rely on LLM-as-a-judge evaluation [17], which is known to exhibit biases including a preference for the judge’s own generations [11], and whose pass rates can vary substantially across repeated runs on the same system [16]. We therefore compare arms only within concurrent runs and report inter-judge agreement.

Table 1: The 2×2 design. A vs. B corresponds to the flat vs. hierarchical comparison of [1].

	No verifier trigger	Verifier trigger
No opinion trigger	A: flat (first draft is final)	C: revise only on runtime errors
Opinion trigger	B: reviewer may send back	D: both

3 Experimental Design

3.1 The team

The team builds small multi-file React applications from Japanese user requests. A planner decides the files; three specialists write CSS and configuration, types and utilities, and React components respectively; a reviewer reads all files and returns, per file, a score (0–100), a status (pass/warn/fail), issues and revision instructions. Files marked as failing are sent back to their author. All agents use `gemini-3.5-flash-lite` with medium thinking, as in production. The generated application is rendered in a sandboxed browser preview.

3.2 Two revision triggers, crossed

We made each trigger independently switchable and crossed them (Table 1). Both triggers were capped at two revision rounds, as in [1].

- **Opinion trigger.** The reviewer reads the code and decides whether to send files back. The production reviewer additionally runs a deterministic CSS/class-name consistency check; we disabled it so that the decision rests on the model’s judgment alone.
- **Verifier trigger.** The application is executed in the preview; if runtime errors appear, the errors are sent back with the instruction to make the app work. If no error appears, nothing happens.

3.3 Calibrating the manager

The production reviewer is lenient: with its acceptance threshold of 75 it sent work back in only 6 of 96 runs in our previous measurement, and in a pilot with the threshold raised to 95 it did so in 1 of 12. Under such leniency, opinion authority would almost never be exercised and could not be studied. The manager in [1] followed an explicit decision guide and looped in 32 of 43 products. To make opinion authority actually operate, we instructed the reviewer in arms B and D to list concrete improvements per file in the first round (at most three) and sent back every file for which it listed any; in later rounds files were sent back if the score fell below 95. This is a deliberate manipulation, not the production setting, and we return to it in Section 7.

3.4 Tasks, runs and measures

- **Tasks.** 24 small applications (calculator, to-do list, kanban, snake game, charts, a 3D solar system, a mortgage calculator, etc.), each requested twice per arm: 48 runs per arm, 192 in total. The four arms ran on four servers at the same time (2026-10-15 23:55 to 2026-10-16 01:00 JST) to equalize model-side conditions. One run in arm A lost its connection to the model API (ECONNRESET) and was rerun under the same arm.
- **Runs.** A run is “working” if the preview appears, shows no error, is not blank, and survives a click on its first button.

Table 2: Quality by arm (48 runs each). Rubric scores are the mean of two judges.

	A (none)	B (opinion)	C (verifier)	D (both)
Working runs	47/48	47/48	48/48	48/48
Pass (judge = 2)	77.1%	81.3%	83.3%	87.5%
Overall score	4.64	4.69	4.67	4.78
Product pillar	4.57	4.68	4.59	4.77
Code pillar	4.75	4.72	4.79	4.78
Fulfillment	4.66	4.84	4.75	4.91
Visual	4.53	4.59	4.42	4.64
Usability	4.53	4.59	4.61	4.78
Clarity	4.77	4.88	4.82	4.92
Conciseness	4.72	4.56	4.75	4.65

- **Pass judge.** `gemini-3.8-flash` scores each final screenshot plus code 0/1/2 three times; the median is used and 2 counts as a pass.
- **Rubric judges.** Following [1], two judges (`gemini-3.8-flash` and `gemini-3.7-flash`) rate five dimensions on 1–5 after writing qualitative notes: fulfillment, visual quality, usability (together the *product* pillar), code clarity and code conciseness (the *code* pillar). The overall score is the mean of the five; we average the two judges. Judges were blind to the arm.
- **Code content.** From the event stream we recover each file’s first draft and final version and count non-comment lines and *defensive constructs*—a fixed lexicon (`try`, `?.`, `??`, early returns on falsy values, `typeof...===‘undefined’`, `Array.isArray`, `|| []`, etc.)—as an analogue of the hedging lexicon in [1].
- **Cost.** Calls and input/output/thinking tokens per role, priced at the model’s list price (\$0.30 per million input tokens, \$2.50 per million output and thinking tokens).
- **Statistics.** The unit is the (task, run) pair ($n = 48$). Pass rates are compared with exact McNemar tests, scores with paired t and Wilcoxon tests. Main effects (mean of the two arms with a trigger minus the mean of the two without) and the interaction are computed per pair, with 95% intervals from a cluster bootstrap over tasks (4,000 resamples). We also fit a linear mixed model with a random intercept for task and a logistic model with task-clustered errors. Holm correction [5] is applied to the twelve primary comparisons.

4 Results

4.1 Quality did not change

Table 2 shows the four arms. Pass rate and overall score increase slightly from A to D, but no contrast is significant (Table 3). The comparison that mirrors the original study, A vs. B, gave 7 pairs where only A passed and 9 where only B passed ($p = 0.80$) and an overall-score difference of +0.05 ($d_z = 0.06$, $p = 0.68$). The mixed model agrees (opinion +0.05, $p = 0.61$; verifier +0.03, $p = 0.78$; interaction +0.05, $p = 0.71$), as does the logistic model for passing (all $p > 0.3$). All Holm-adjusted p values for the primary comparisons are 1.0.

At the dimension level, opinion authority moved product scores and clarity upward and *conciseness downward* (−0.13, 95% CI −0.34 to +0.08), consistently in both comparisons (A→B −0.16, Wilcoxon $p = 0.098$; C→D −0.10). Conciseness was also among the clearest effects in [1] ($d = 0.44$). None of these dimension-level effects is significant here. The lower end of the overall-score interval

Table 3: Main effects and interaction with 95% cluster-bootstrap intervals. All intervals include zero.

Effect	Pass (pt)	Overall	Product	Code
Opinion	+4.2 [−6.2, +13.5]	+0.08 [−0.09, +0.25]	+0.14 [−0.04, +0.33]	−0.02 [−0.20, +0.15]
Verifier	+6.2 [−3.1, +15.6]	+0.06 [−0.10, +0.20]	+0.06 [−0.14, +0.25]	+0.05 [−0.05, +0.16]
Interaction	±0.0 [−14.6, +16.7]	+0.05 [−0.20, +0.31]	+0.08 [−0.24, +0.39]	+0.02 [−0.20, +0.23]

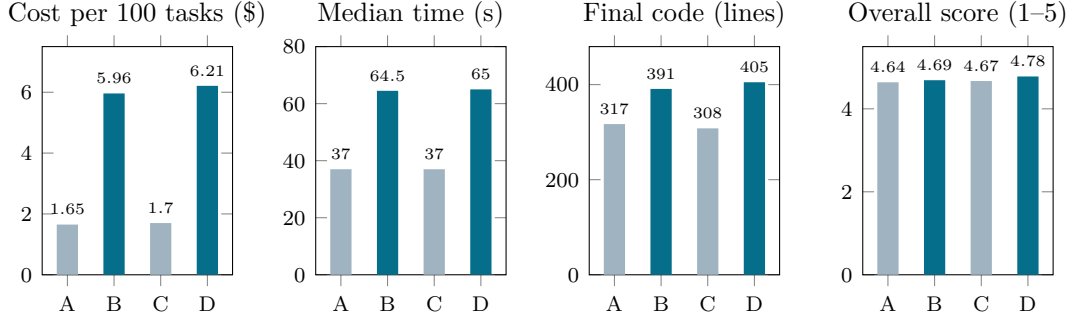


Figure 1: Opinion authority (dark) multiplied cost by 3.6, latency by 1.7 and code size by 1.27, while the overall score barely moved.

(−0.09) is about the size of the original paper’s effects (≈ 0.09 points on Clarity and Utility), so a degradation of that magnitude cannot be excluded; larger ones are unlikely.

4.2 What did change: cost, latency, code size

Opinion authority triggered 73 revision rounds in each of B and D (one round in 23 pairs, two in 25). It raised calls from 4.8 to 11.3 per task, tokens from 13.7k to 56.3k (4.1 $\times$), cost per 100 tasks from \$1.65 to \$5.96 (main effect +\$4.4, 95% CI +4.0 to +4.8), median time from 37s to 64.5s (+28.7s, +23.3 to +33.4) and final code from 317 to 391 lines (+85.6, +63.9 to +108.2); all $p < 0.001$ (Figure 1). The verifier changed none of these appreciably (cost +\$0.2, time +0.5s, lines +2.0; all $p > 0.4$).

4.3 Drafts are the same; revision inflates them

As in [1], the first drafts in the opinion arm were indistinguishable from the flat arm’s final output: 324 vs. 317 lines ($p = 0.57$) and 0.41 vs. 0.50 defensive constructs per 100 lines ($p = 0.23$). Within B, revision grew the code from 324 to 391 lines (+21%, $d_z = 1.15$, $p < 0.001$) and the density of defensive constructs from 0.41 to 0.59 per 100 lines (+42%, $d_z = 0.45$, $p = 0.003$); in D, density rose from 0.52 to 0.66 (+28%, $d_z = 0.46$, $p = 0.003$). On average 82% of the first draft survived into the final version (88% in the original study). The between-arm difference in final density (B vs. A) was not significant (+0.09, $p = 0.20$) because chance differences between first drafts offset it; the robust finding is the within-run growth.

4.4 What the reviewer asked for

The reviewer wrote 910 critiques in B and D; in the first round it listed 8.4 (B) and 8.9 (D) per run. We classified every critique into six categories with `gemini-3.8-flash` and independently

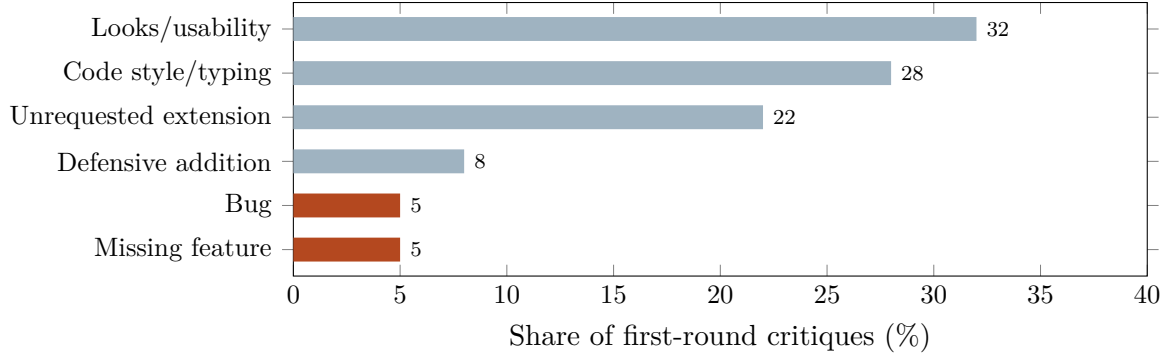


Figure 2: Content of the reviewer’s first-round critiques ($n = 832$). Only bugs and missing features (orange) are checkable by running the app.

with **gemin-3.7-flash** (agreement 93.4%, $\kappa = 0.91$; on the binary split “runtime-checkable or not” 95.9%, $\kappa = 0.81$). Of the 832 first-round critiques, 32% concerned looks and usability, 28% code style or typing, 22% unrequested extensions (“no extensibility for dark mode”), 8% defensive additions for situations that do not occur in use (“no empty state”), and only 5% bugs and 5% missing requested features (Figure 2). The second classifier gave 11% for bugs plus missing features. Thus about 90% of what the opinion-only manager demanded could not be confirmed by running the application.

In later rounds the share of bugs rose (34% of 61 second-round and 29% of 17 third-round critiques), suggesting that first-round revisions sometimes introduced defects. In one run (an employee table), the reviewer sent back all four files with requests about a “dated” sort indicator, dark-mode spacing and an “overly minimal” interface; the revised application failed at runtime, and the reviewer then passed it with a score of 95. A manager that cannot execute the code cannot see that its own requests broke it.

4.5 The verifier fired only when needed

The verifier fired in 3 of 48 runs in C (four repair rounds) and in none in D. All three repaired runs ended working and passing. Because the team rarely produces broken applications (47/48 working in A), the verifier’s effect on quality cannot be estimated precisely (+6.2 points, $p = 0.16$), but its cost was negligible: comparing A and C, cost per 100 tasks was \$1.65 vs. \$1.70 and median time 37s in both, while working runs rose from 47 to 48.

4.6 Dose–response

The original study found that each revision loop lowered clarity by 0.14 points. We found no such decline: within B, the number of opinion loops was unrelated to the overall score ($\rho = 0.07$, $p = 0.64$), and within D runs with two loops scored slightly higher (pass-judge slope +0.19, $p = 0.036$). Because the second loop is triggered by a low score, loop count is chosen by the manager and this relation is not causal.

4.7 Robustness

Inter-judge agreement was high (Krippendorff’s α [9] = 0.93 for the overall score; 0.76–0.93 per dimension), far above the 0.08–0.30 reported in [1], though both judges belong to the same model

Table 4: Replication with **gemini-3.8-flash** as every agent (48 runs per arm).

	A (none)	B (opinion)	C (verifier)	D (both)
Working runs	42/48	44/48	47/48	47/48
Pass (judge = 2)	83.3%	79.2%	79.2%	93.8%
Overall score	4.49	4.59	4.46	4.79
Product / Code pillar	4.40 / 4.63	4.50 / 4.71	4.37 / 4.61	4.78 / 4.81
Revision rounds (opinion / verifier)	0 / 0	76 / 0	0 / 7	74 / 3
Cost per 100 tasks	\$5.77	\$22.49	\$6.27	\$22.60
Median time	103 s	234 s	99 s	236 s
Final code (lines)	862	1029	854	1064

family. Leaving out either judge leaves the A–B comparison unchanged ($+0.00$, $p = 0.98$ with **gemini-3.8-flash** only; $+0.10$, $p = 0.43$ with **gemini-3.7-flash** only). With 48 paired units the design detects $d_z \approx 0.41$ with 80% power, comparable to the original study’s effect sizes and sample (43 pairs). Scores are near ceiling (4.6–4.8), as in the original (4.4–4.7).

4.8 Blinded human check

Following the original study’s human check, the author compared one pair of applications per task (24 pairs, flat arm A vs. opinion arm B, same task and run; the run was chosen at random among runs in which both arms produced an application). The two applications were shown side by side as live, interactive previews built by the production renderer, with arm labels hidden and left/right order randomized; the rater operated both before choosing on a five-point scale (-2 to $+2$, “which is better for the person who asked”) and wrote a one-line reason. The rater did not see the code or any judge score.

The human rater preferred B in 11 pairs, A in 8, and rated 5 as equal; the mean preference was $+0.21$ toward B (bootstrap 95% CI -0.21 to $+0.62$; Wilcoxon $p = 0.31$; sign test $p = 0.65$). Excluding the one pair in which A’s application failed with a runtime error, the split was 10 vs. 8 ($+0.13$, $p = 0.51$). On the same pairs the LLM judges’ mean difference was $+0.05$ ($p = 0.85$). Human preferences correlated moderately with the judges’ score differences (Spearman $\rho = 0.48$, $p = 0.017$; $\rho = 0.40$, $p = 0.057$ without the failed pair), but the direction agreed in only 12 of 24 pairs, so the judges are informative in aggregate rather than per item.

The stated reasons show what a user notices. Of the 19 non-tied judgments, 12 cited a feature present in only one version—a break timer, an undo button, example sentences, a map, a payment-schedule comparison—and these went both ways (7 favouring B, 5 favouring A); 4 cited layout or legibility and 3 a defect (a crash, mixed-script text in a quiz, drag-and-drop that did not work). We did not determine whether the features that favoured B were added during revision or already present in the draft.

5 Replication with a Stronger Model

We repeated the full 2×2 experiment with every agent running **gemini-3.8-flash** (same 24 tasks $\times$ 2 runs, same manager calibration, four concurrent arms; 192 generations, 2026-10-16 17:32–21:20 JST). Costs are priced at that model’s list price (\$0.75 / \$3.75 per million input / output tokens). Because **gemini-3.8-flash** is also a judge, we report each judge separately in the robustness check.

Quality. Again no contrast survived correction (Table 4). The main effects were opinion +0.22 on the overall score (95% CI -0.01 to $+0.45$, $p = 0.09$) and verifier +0.09 (-0.13 to $+0.31$, $p = 0.47$). Arm D tended to be best (pass 93.8% vs. 79.2–83.3%; $C \rightarrow D$ $p = 0.09$, $B \rightarrow D$ $p = 0.07$; interaction on pass +18.8 points, $p = 0.10$), but none of these is significant after Holm correction, and the mixed model gives opinion +0.10 ($p = 0.58$), verifier -0.03 ($p = 0.89$) and interaction +0.23 ($p = 0.37$). As before, opinion authority did not lower quality.

Cost and code. The cost of opinion authority replicated and grew: $3.9\times$ cost (main effect +\$16.5 per 100 tasks, 95% CI $+15.1$ to $+18.0$), +139s per task ($+124$ to $+153$), and +188 lines of code ($+147$ to $+236$), all $p < 0.001$. The verifier changed none of these ($p > 0.3$).

Drafts and inflation. First drafts in B were again indistinguishable from A (880 vs. 862 lines, $p = 0.58$; defensive density 0.64 vs. 0.70, $p = 0.55$). Revision grew the code by 17% ($d_z = 1.06$, $p < 0.001$) and the density of defensive constructs by 52% (0.64 to 0.97 per 100 lines, $d_z = 0.52$, $p = 0.001$; in D +16%, $p = 0.04$). With this model the between-arm difference in final density was also significant (opinion main effect +0.20, $p = 0.04$). 81% of the first draft survived.

What the reviewer asked for. The stronger reviewer wrote more critiques (1,451 in the first round, about 15 per run). Classified with `gemin-3.8-flash`, 34% concerned code style, 33% looks and usability, 10% defensive additions, 8% unrequested extensions, 10% bugs and 4% missing features: 14% runtime-checkable, compared with 10% for the weaker reviewer. In later rounds bugs again rose (24% and 22%).

Verifier. `gemin-3.8-flash` produced broken applications more often (42/48 working in A vs. 47/48 with the weaker model), so the verifier fired more: in 4 runs of C (7 rounds) and 2 runs of D (3 rounds). Working runs rose from 42 (A) to 47 (C), but judged quality did not (pass 83.3% vs. 79.2%), because the judges also penalized visual and functional shortcomings that do not raise runtime errors.

Dose–response. Unlike the first experiment, more opinion loops went with lower scores, as in [1]: within D, -0.39 overall points per loop ($p = 0.001$); within B, -0.53 ($p = 0.055$). Because a second loop is triggered only when the manager scores the revised draft below 95, harder or weaker drafts receive more loops, and this association cannot be read as a causal effect of revision.

Robustness. Agreement between the two rubric judges was even higher ($\alpha = 0.99$ overall; 0.90–0.99 per dimension). The A–B difference was the same with either judge alone ($+0.10$, $p = 0.59$), giving no sign that the `gemin-3.8-flash` judge favoured its own generations differently across arms.

6 Discussion

What replicates and what does not. The mechanism replicates. Loop-back authority did not change first drafts; differences arose in revision, which added material and precaution—hedged in prose, defensive constructs in code—and about nine in ten of the manager’s requests were not checkable by execution. The outcome did not replicate: judged quality did not fall. Two explanations are plausible. Much of what revision adds to code (responsive layout, empty states,

stricter types) is harmless to a user, whereas hedging in a report blunts the reader’s judgment. And code carries a strong external constraint—the application must still run—that prose lacks.

Liability in cost rather than quality. In our setting, unverifiable authority was paid for not in judged quality but in money ($3.6\times$), latency ($1.7\times$) and code size ($1.27\times$). When quality does not change, a 3.6-fold cost is itself a liability. The conjecture of [1] is therefore not supported on quality alone but is strongly supported on cost-effectiveness: verifier-triggered authority repaired every broken app it saw at almost no cost, whereas opinion-triggered authority spent heavily without demonstrably repairing anything checkable.

Across models. With a stronger model the same pattern held, more sharply: the cost of opinion authority rose to $3.9\times$, inflation of defensive code became significant between arms, and the loop count was negatively associated with quality as in the original paper—though, by design, that association is confounded with draft difficulty. The arm that combined a verifier with an opinion reviewer tended to be best with the stronger model; whether verification makes opinion authority *useful*, rather than merely harmless, is the most promising question for a larger study.

Design implication. Revision should be triggered by verifiable signals—runtime errors, failing checks, missing requested features—rather than by a reviewer’s unverifiable preferences, consistent with the self-correction literature [4, 7]. A lenient reviewer that rarely loops, as in our production system, is a reasonable default.

7 Limitations

- **Manager calibration.** To make opinion authority operate, the reviewer was required to list improvements in the first round and every listed file was sent back. This differs from both our production reviewer and the decision-guided manager of [1]; a naturally looping manager might behave differently.
- **Few verifier events.** The verifier fired in only 3 of 48 runs, so its effect and the interaction are poorly estimated. Harder tasks or weaker workers would be needed.
- **Limited human evaluation.** Both rubric judges and the pass judge are Gemini models, and high agreement may reflect shared preferences. The human check (Section 4.8) covers 24 A–B pairs of the first experiment with a single rater, who is also the author and the system’s developer; it was blinded to arm but has no inter-rater agreement, and C and D were not compared by a human.
- **One team, one model family.** Within each experiment all agents used a single model; the two experiments used two models of the same family (Section 5), whereas the original study mixed five model families. The replication’s critique classification used a single classifier.
- **Task scope.** 24 small Japanese-language applications; larger codebases or longer tasks may differ.
- **Defensive-construct lexicon.** The fixed lexicon does not distinguish necessary from superfluous precautions, mirroring the limitation of the hedge lexicon in the original work.
- **Critique classification.** Both classifiers are LLMs from the same family; no human coding.
- **Loop count.** The second loop is triggered by the manager’s score, so dose–response estimates are correlational.

8 Conclusion

We tested, in a production code-generation team, the conjecture that loop-back authority pays only when it can verify. Neither opinion-triggered nor verifier-triggered revision changed judged quality beyond chance, and unlike the original study opinion authority did not lower it. But opinion authority multiplied cost by 3.6, latency by 1.7 and code size by 1.27; revision inflated code with defensive precautions; and nine in ten of the manager’s requests were not checkable by running the application. A verifier that ran the code fired only in the three runs that needed it, repaired all of them, and cost almost nothing. A replication with a stronger model reproduced these results, with the cost penalty growing to $3.9\times$. In this setting, authority without verification does not so much break the work as spend money and time without changing it. Revision should be tied to problems that can be verified.

Use of AI assistants. The author used AI assistants for code development, experiment orchestration, and drafting of this manuscript, and takes full responsibility for its content.

Data and code. Tasks, per-run results, generated applications and screenshots, full event streams, judge outputs, critique labels, the human ratings with the blinding key, and all analysis scripts are available at <https://github.com/YUALAB/loop-back-authority-verify> and archived at <https://doi.org/10.5281/zenodo.23423127>. Re-running the analysis script on the released data reproduces every statistic reported here. The generation scripts are released as a record of the procedure; they require the production application, which is not public. A Japanese version of this report is available at <https://www.aquallc.jp/research/authority-needs-verification/>.

References

- [1] Burak Agachan, Max van Duijn, and Amirhossein Zohrehvand. Loop-back authority in LLM agent teams: A paired experiment on flat and hierarchical coordination. *arXiv preprint arXiv:2609.14767*, 2026.
- [2] Mert Cemri, Melissa Z. Pan, Shuyi Yang, et al. Why do multi-agent LLM systems fail? *arXiv preprint arXiv:2503.13657*, 2025.
- [3] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2304.05128.
- [4] Zhibin Gou, Zhihong Shao, Yeyun Gong, et al. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2305.11738.
- [5] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.
- [6] Sirui Hong, Mingchen Zhuge, Jonathan Chen, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2308.00352.

- [7] Jie Huang, Xinyun Chen, Swaroop Mishra, et al. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.01798.
- [8] Prathyusha Jwalapuram, Hehai Lin, Chuyuan Li, et al. The illusion of multi-agent advantage. *arXiv preprint arXiv:2606.13003*, 2026.
- [9] Klaus Krippendorff. *Content Analysis: An Introduction to Its Methodology*. SAGE, 4th edition, 2018.
- [10] Aman Madaan, Niket Tandon, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2303.17651.
- [11] Arjun Panickssery, Samuel R. Bowman, and Shi Feng. LLM evaluators recognize and favor their own generations. *arXiv preprint arXiv:2404.13076*, 2024.
- [12] Chen Qian, Wei Liu, Hongzhang Liu, et al. ChatDev: Communicative agents for software development. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2307.07924.
- [13] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2303.11366.
- [14] Jiawei Xu, Arief Koesdwiady, Sisong Bei, et al. Rethinking the value of multi-agent workflow: A strong single agent baseline. *arXiv preprint arXiv:2601.12307*, 2026.
- [15] Shinji Yumura. Where does an agent team fail? applying the MAST taxonomy to a production code-generation team. AQUA Research Technical Report 005, AQUA LLC, 2026. In Japanese. <https://www.aqualic.jp/research/where-agent-teams-fail/>.
- [16] Shinji Yumura. Can an LLM judge tell whether generated apps got better? AQUA Research Technical Report 002, AQUA LLC, 2026. In Japanese. <https://www.aqualic.jp/research/judging-generated-apps/>.
- [17] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, et al. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *NeurIPS Datasets and Benchmarks Track*, 2023. arXiv:2306.05685.