FEEDBACK ALIGNMENT IN SELF-IMPROVING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-improving agents often use the same evaluation signal both to select updates and to report progress. When that signal captures only part of the intended objective, a rising score does not necessarily mean better task performance: the agent may instead be getting better at satisfying the evaluator. We study this failure mode in scaffold-level self-improvement, where a fixed language model repeatedly revises the executable program used to solve future tasks. We introduce a controlled protocol that separates visible feedback V used for scaffold selection from held-out target evaluation T used only for reporting. Across two-voice counterpoint repair and cross-database SQL repair, scaffold evolution can raise the visible score while lowering the independently measured target. Ordinary best-of-five selection also reduces the counterpoint target, but less than the full evolution procedure under the same call cap. Retained scaffolds further encode evaluator-specific prompting, checking, and selection rules. These results indicate that a rising optimization score is insufficient evidence of improvement; in the tested comparisons, feedback alignment predicts the direction of target change.

1 INTRODUCTION

Language-model agents rely on more than the model itself. They also use prompts, tools, memory, validation rules, and code that controls when and how the model is called. We refer to this surrounding program as the agent's *scaffold*. Recent systems improve a scaffold over several rounds: they revise prompts from critiques, search over agent programs, and keep successful code for later tasks (Madaan et al., 2023; Fernando et al., 2024; Romera-Paredes et al., 2024; Hu et al., 2024; Novikov et al., 2025; Zhang et al., 2025). The model weights stay fixed, while the program that uses the model changes.

These systems need feedback to decide which change to keep. Some tasks provide exact feedback: code can be compiled, proofs can be checked, and algorithms can be tested against a precise specification. Open-ended tasks usually rely on preference models, AI feedback, model judges, or sets of domain-specific metrics (Ouyang et al., 2022; Bai et al., 2022; Zheng et al., 2023). Such feedback can be useful while still ranking candidates differently from the intended objective.

The same feedback score often reports progress and selects the scaffold for the next round. A higher score may therefore reflect better task performance or a better strategy for satisfying the evaluator. Goodhart effects and reward-model overoptimization establish this distinction for policies and individual outputs (Manheim & Garrabrant, 2018; Skalse et al., 2022; Gao et al., 2023). Scaffold evolution can encode the resulting strategy in a program and reuse it on later tasks. Existing work seldom evaluates revised scaffolds against a separate target or compares scaffold rewriting with ordinary best-of-N sampling.

We study this problem with two-voice counterpoint repair. The input contains a fixed lower voice and a corrupted upper voice. Many different repairs can satisfy the formal rules, including consonance, forbidden parallel motion, voice crossing, melodic motion, and cadence treatment (Fux, 1725). The held-out instances and their target scores never enter the improvement loop. Selection instead uses either the same graded rule criterion on search data or an incomplete proxy made from simple musical statistics.

Figure 1 summarizes the experiment and its three outcome measures. Within each paired comparison, the task, starting scaffold, model, search procedure, data split, and resource limits remain fixed; only the selection evaluator changes. We assess directional agreement between visible-score ascent

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

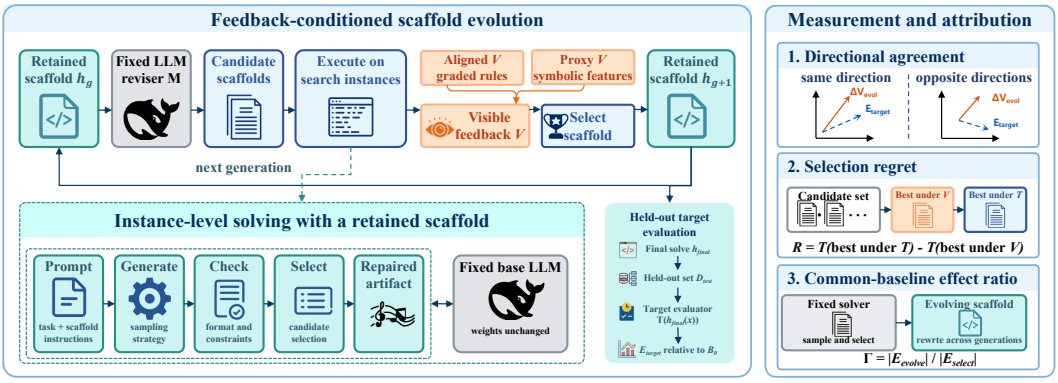


Figure 1: Overview of the experiment. Visible feedback V selects scaffold revisions, whereas target T is evaluated only on held-out D_{test} after evolution. The right panel summarizes the three outcome and attribution measures.

and the final held-out target effect, measure selection regret in archived SQL candidate pools, and compare scaffold evolution with a fixed solver through a descriptive common-baseline effect ratio. SQL repair repeats the aligned-versus-proxy comparison using result-set agreement as the target and string-level style features as the proxy. We also audit the retained source code.

The counterpoint proxy condition provides the clearest separation: its visible score rises by 0.098, while its final scaffold has a target effect of -0.292 relative to B0 ($p_{\text{Holm}} = 0.007$). The graded-rule condition has a target effect of $+0.032$ but does not clear family-wise correction. Against the same B0 baseline, best-of-five proxy selection with a fixed solver has an effect of -0.070. SQL supplies the same directional contrast: execution-guided evolution has a result-set effect of $+0.117$, whereas style-guided evolution has an effect of -0.119. The program audits document how the selected evaluator is retained in prompts, scorer calls, checks, and candidate-selection rules.

2 Related Work

2.1 From iterative refinement to persistent agent self-improvement

Language-model systems improve at several different levels. Output-level methods repeatedly critique and revise a current answer: Self-Refine uses model-generated feedback for iterative revision, Reflexion stores verbal feedback in episodic memory, OPRO conditions an optimizer on previous solution–score pairs, and TextGrad propagates natural-language critiques through compound systems (Madaan et al., 2023; Shinn et al., 2023; Yang et al., 2024; Yuksekgonul et al., 2024). These methods establish that feedback can guide additional test-time computation, with optimization applied to the current response or a bounded textual state. Intrinsic correction can also fail when the loop lacks a reliable external signal (Huang et al., 2024a). Our setting commits an executable program after each round, so a feedback-induced strategy changes how future instances are prompted, checked, and selected.

Prompt and program optimization make this persistence explicit. Promptbreeder and EvoPrompt evolve instructions, DSPy compiles modular pipelines against a metric, and STOP rewrites an LM-infused optimizer while holding the base model fixed (Fernando et al., 2024; Guo et al., 2024; Khattab et al., 2023; Zelikman et al., 2023). FunSearch and AlphaEvolve couple LLM proposals with automated evaluators to evolve executable algorithms, while ADAS searches over agent implementations (Romera-Paredes et al., 2024; Novikov et al., 2025; Hu et al., 2024). More directly self-referential systems edit the agent itself: SICA modifies its own coding-agent implementation, and the Darwin Gödel Machine maintains an open-ended archive of empirically evaluated agent variants (Robeyns et al., 2025; Zhang et al., 2025). These systems demonstrate that evaluator-guided search can discover stronger procedures and generally treat the supplied evaluator as the objective. Our study examines the procedures retained when that evaluator and an independent target induce different rankings.

The classical Gödel machine justifies self-modification through a proof that the modification improves a formally specified utility (Schmidhuber, 2003). Contemporary scaffold evolution relies on finite empirical evaluations, so its behavior depends on the information supplied by feedback. We measure whether retained scaffold changes improve an independent target, how much target score is lost during selection, and how the effect of the full evolution procedure compares with ordinary sampling.

2.2 Feedback misspecification and independently verifiable targets

Goodhart effects arise when a measure becomes an optimization target, and reward hacking formalizes the distinction between an observed proxy and the objective it was intended to represent (Manheim & Garrabrant, 2018; Skalse et al., 2022). AI Safety Gridworlds exposes specification failures through separate observed reward and hidden performance functions, while reward-model overoptimization shows that stronger optimization of a learned reward can eventually reduce performance under a stronger reference measure (Leike et al., 2017; Gao et al., 2023). We apply the observed-versus-hidden separation to executable scaffolds that persist across rounds. This introduces a second source of pressure beyond selecting a high-scoring response, because the selected program can encode evaluator-specific behavior and apply it to subsequent instances.

Automatic evaluators make this issue operational for language-model agents. Preference models, AI feedback, LLM judges, and composite metrics provide scalable supervision (Ouyang et al., 2022; Bai et al., 2022; Zheng et al., 2023), yet evaluator judgments can reflect presentation and other incidental properties, and bias can remain even in agent-based evaluation protocols (Chen et al., 2025). Static agreement is insufficient once outputs are selected adaptively: the relevant question is whether the evaluator preserves the target ordering within candidate distributions created by the optimizer. Our selection-regret measure directly tests that ordering, while a call-cap-matched control compares ordinary best-of-N selection with persistent scaffold revision.

Formal symbolic domains make this separation measurable while allowing multiple valid outputs. Constraint programming has long encoded counterpoint, harmony, rhythm, and form as explicit relations (Anders & Miranda, 2011). DeepBach conditions polyphonic generation on positional constraints, RL-Tuner combines a learned sequence model with music-theory rewards, and recent diffusion methods incorporate non-differentiable or fine-grained symbolic guidance (Hadjeres et al., 2017; Jaques et al., 2016; Huang et al., 2024b; Zhu et al., 2025). Those works use constraints to improve music generation. In our study, counterpoint rules define the held-out target, while incomplete but locally plausible musical statistics guide the proxy condition. SQL repair supplies a second formal target through execution, allowing the comparison to be repeated with a different artifact and failure mode.

3 Feedback-Conditioned Scaffold Evolution

In a self-improvement loop, feedback affects future behavior by deciding which solver program is retained. We distinguish the retained scaffold, the evaluator used for selection, and target measurements computed on held-out instances.

3.1 Problem formulation

Let h_g be the solver program retained at generation g. A fixed language model proposes up to $K = 2$ revisions per generation, and C_g denotes the subset that passes program validation. The visible feedback function V selects the program used in the next generation:

$$h_{g+1} = \arg\max_{h \in C_g} \mathbb{E}_{x \sim D_{\text{search}}}[V(h(x))]. \tag{1}$$

Equation 1 directly rewards higher V, while the independently measured target can increase or decrease. The primary target endpoint compares the final scaffold with the single-shot baseline B0 on D_{test} (Section 4); per-generation target scores are retained only for descriptive trajectories. Held-out instances and their scores are unavailable during proposal and selection. The experiment separates two forms of independence. First, selection and reporting always use disjoint data: V is aggregated on D_{search}, whereas T is evaluated on D_{test}. Second, the proxy conditions also separate

the scoring functions themselves. In the aligned counterpoint condition, V_{rule} and T_{rule} are the same graded-rule function applied to different instances; in the proxy condition, $V_{\text{proxy}} \neq T_{\text{rule}}$. Thus the aligned condition is a search-to-test positive control, not a comparison between two independent scoring rules. Here, *self-improvement* means that retained code changes future prompting, sampling, checking, and selection. Model weights and the outer search procedure remain fixed.

3.2 Feedback-conditioned scaffold update

The scaffold implements a task-level function $h_g : x \mapsto y$. It may build prompts, allocate model calls, generate several outputs, call the visible scorer, check formats, select an output, and apply fallback rules. Let $\mathcal{U}$ be the fixed procedure that proposes revisions, and let H_g contain the retained program, its outputs, and its visible scores. One generation is

$$C_g = \mathcal{U}(h_g, H_g; M), \qquad h_{g+1} = \text{Select}_V(C_g, D_{\text{search}}), \tag{2}$$

where M is the fixed model that revises the code. Candidate programs must pass syntax checks and run successfully in a sandbox. Each valid program then solves the same search instances under the same resource limits. We average its visible feedback and retain the highest-scoring program. Its source code, outputs, and scores are added to H_{g+1}, so a strategy found in one generation can affect later generations.

There are two levels of selection. At the *scaffold level*, the outer loop compares up to $K = 2$ valid revised programs and retains one for the next generation. We fixed $K = 2$ to keep the cost of executing candidate programs tractable across all conditions; it is a resource setting, not a claimed optimum, and we do not vary it. At the *instance level*, a scaffold may spend up to $N = 5$ base-model calls while producing and selecting a repair for one task instance. The value $N = 5$ is a per-instance call cap and is unrelated to K. A proxy can affect the current output and can also change future prompts, scorer calls, checks, and fallback rules. The fixed-solver control preserves instance-level selection while disabling scaffold-level revision.

3.3 Outcome and attribution measures

Directional agreement. We compare the sign of visible-score ascent during evolution with the sign of the final held-out target effect. As formalized in Section 4, these quantities have different prespecified references: visible ascent compares the final and starting scaffolds, whereas the target effect compares the final scaffold with B0. They also use different data and, in the proxy conditions, different scoring functions. In the aligned condition the scoring function is shared, so equality of visible and target scores on the same artifacts would hold by definition; the informative question is instead whether selection on D_{search} transfers to a positive effect on D_{test}. We therefore use directional agreement as an endpoint summary and do not treat coincident aligned-score trajectories as an empirical result.

Selection regret. We re-score an archived SQL candidate pool on one common evaluation set D_{eval}. Let

$$\begin{aligned} h_V^* &= \arg\max_{h \in C} \mathbb{E}_{x \sim D_{\text{eval}}}[V(h(x))], \\ h_T^* &= \arg\max_{h \in C} \mathbb{E}_{x \sim D_{\text{eval}}}[T(h(x))], \\ R &= \mathbb{E}_{x \sim D_{\text{eval}}}[T(h_T^*(x)) - T(h_V^*(x))]. \end{aligned} \tag{3}$$

When $V = T$ on this common evaluation set, regret is zero by construction. We report both pooled regret over the full archived pool and within-generation regret over each generation's candidates. Positive regret is the target score lost by choosing the candidate preferred by V.

Common-baseline effect ratio. Let E_{select} and E_{evolve} denote the target effects of fixed-solver best-of-five selection and scaffold evolution, both measured against the same single-shot baseline B0. We compare their magnitudes with

$$\Gamma = \frac{|E_{\text{evolve}}|}{|E_{\text{select}}|}. \tag{4}$$

A value above one means that the full evolution procedure has a larger effect relative to B0 than candidate selection with a fixed solver. We report the ratio only when all four fixed-solver repetitions give a denominator with the same sign. It is a descriptive common-baseline ratio, not the direct difference between the systems, a decomposition of outer-search mechanisms, or a quantity used for hypothesis testing.

3.4 Intervention and controls

Only the selection evaluator V changes across the primary paired conditions. The aligned arm ranks candidate scaffolds by graded rule satisfaction on D_{search}; the proxy arm ranks the same candidate programs by surface musical statistics on the same search instances. Both arms are evaluated afterward by graded rule satisfaction on D_{test}. The task, starting scaffold, data splits, model, proposal procedure, retention rule, sandbox, number of generations, and resource caps stay fixed. This intervention therefore asks whether replacing the candidate ranking—rather than the task or optimizer—changes the held-out target effect.

The loop cannot access held-out examples or their target scores, and the proxy arm cannot call the target evaluator during execution or selection. An independent rerun further withholds the score and error report from the proposal prompt while retaining proxy-based outer selection. The fixed-solver control measures best-of-N selection with no scaffold rewriting, separating instance-level candidate selection from the full cross-generation procedure. Finally, source-code audits record whether selected programs retain feedback-specific prompting, checking, scorer calls, and candidate-selection rules.

4 Experimental Setup

We instantiate the comparison in two domains with independently computable targets. Counterpoint repair is the primary domain; SQL repair tests the same aligned-versus-proxy contrast with a different artifact and verifier.

4.1 Formal counterpoint repair

Each instance x contains a fixed cantus firmus and a corrupted upper voice in MusicXML (Good, 2001). Clean voices are obtained by bounded backtracking under the evaluation rules, after which violations are injected at three logical difficulty levels. A repair y preserves the cantus firmus and rhythmic grid. For rule indicators $r_j(y; x) \in [0, 1]$ and weights fixed before any run, the target is $T_{\text{music}}(y; x) = \sum_j w_j r_j(y; x) / \sum_j w_j$. It covers consonance, forbidden parallels, voice crossing and overlap, melodic intervals, range, repeated tones, motion, and cadence. This is a formal rule-satisfaction target and admits many valid note sequences; it is not a measure of general musical quality.

Rule-aligned feedback uses the graded rule score on search instances. The symbolic proxy is a fixed weighted combination of consonance rate, interval variety, motion balance, pitch variety, step ratio, key fit, leap softness, and range fit. These locally plausible summaries omit temporal and relational prohibitions; 40% of their weight was classified as potentially gameable before any run. A 400-step local optimizer with no model calls raises the proxy to 0.910 while rule satisfaction falls from 0.856 to 0.605. Full calibration appears in Appendix A.

A learned ridge proxy provides an additional pre-evolution calibration: despite fitting ordinary samples well (MAE 0.092, Spearman $\rho = 0.824$), local search raises its prediction to 0.992 while rule satisfaction falls from 0.856 to 0.424. The main evolution comparison uses only graded rules and the symbolic proxy.

Music examples are divided before evolution into search, validation, and held-out partitions; the held-out partition is used only for reporting. Difficulty is determined by the logical reach of the injected violation: local defects, neighborhood-dependent defects, and non-local constraints have initial binary pass rates of 66.7%, 46.7%, and 25.0%. Outputs must parse as MusicXML, preserve the cantus firmus and rhythmic grid, and satisfy the task interface before scoring. Invalid artifacts receive the prespecified floor behavior, without revealing held-out diagnostics to the agent.

4.2 SQL REPAIR

Cross-database text-to-SQL emphasizes task-level generalization (Yu et al., 2018), and execution is a stronger semantic test than string matching (Zhong et al., 2020). We cross 12 query templates with four defect types and four data seeds, yielding 192 instances: 64 each for search, validation, and held-out evaluation. The 64 held-out instances form 48 task groups nested within 12 templates, which define the coarser aggregation levels reported below. For each instance, the candidate and reference queries run on the same generated in-memory database. The aligned evaluator measures graded result-set agreement; the proxy rewards seven non-executing style features.

Each generated defect must remain executable, disagree with the reference result, and be detected by an associated wrong-answer check; otherwise the generator advances to another seed. Execution feedback provides partial credit from the candidate–reference result-set mismatch on that instance. The style proxy inspects only the query string, rewarding properties such as explicit aliases and normalized formatting. Proxy-optimized music remains parseable while accumulating relational-rule violations, whereas proxy-optimized SQL can become polished-looking text that no longer executes correctly.

4.3 EVALUATION METRICS

The two reported changes use different, prespecified references. Let h_{start} and h_{final} denote a condition's starting and last-generation scaffolds, and let b_0 denote the fixed single-shot baseline averaged over four repetitions. On held-out instances we report

$$\begin{aligned} \Delta V_{\mathrm{evol}} &= \mathbb{E}_{x\sim D_{\mathrm{test}}}[V(h_{\mathrm{final}}(x)) - V(h_{\mathrm{start}}(x))], \\ E_{\mathrm{target}} &= \mathbb{E}_{x\sim D_{\mathrm{test}}}[T(h_{\mathrm{final}}(x)) - T(b_0(x))]. \end{aligned} \tag{5}$$

Thus ΔV_{evol} is visible-score ascent during evolution, whereas E_{target} is the final system's target effect relative to B0. They have different reference points and should not be interpreted as two measurements of the same start-to-final contrast. Positive ΔV_{evol} verifies that evolution raised the reported selection metric. In aligned conditions V and T implement the same criterion but remain separated operationally: selection uses non-test data, and target measurement uses held-out instances. In proxy conditions they also measure different properties.

Selection regret R measures the target score lost when visible feedback prefers a different candidate from the target (Equation 3). The amplification ratio Γ compares the absolute target effects of scaffold evolution and the call-cap-matched fixed solver against their common B0 baseline (Equation 4). We additionally report generation-wise trajectories, seed-level directions, invalid-output diagnostics, model calls, and source-code audit counts.

4.4 CONDITIONS AND CONTROLS

Each domain has an aligned condition and a proxy condition. Counterpoint uses graded rule satisfaction or the symbolic music proxy; SQL uses execution agreement or the non-executing style proxy. Within a domain, both conditions start from the same scaffold and use the same model, examples, proposal procedure, number of generations, and resource limits. In the primary counterpoint comparison, every task prompt contains the complete rule list and the exact-rule error report; the intervention changes which evaluator the scaffold can call during improvement. An independent rerun withholds the score and error report from the proposal prompt while retaining the same outer selection criterion. This tests whether selection pressure can reproduce the effect without exposing those diagnostics to the reviser.

The call-cap-matched counterpoint control keeps the initial scaffold fixed, uses exactly five model calls to produce independent candidate repairs, and selects an output using the same visible evaluator. Evolved scaffolds share the five-call cap but may use fewer calls; in the primary proxy condition they average 4.47 calls per instance. The comparison therefore gives the fixed solver at least as much evaluation-time inference as the evolved solver and tests whether best-of-five selection accounts for the observed loss. It does not match the outer-loop cost of proposing and evaluating revised programs. Source-code audits compare the initial and retained scaffolds and record changes to prompts, scorer calls, checks, candidate selectors, and fallback behavior. Held-out instances and their target diagnostics are unavailable to every proposal, execution, and selection step.

Table 1: Two domains instantiate the same separation between visible feedback and an independently evaluated target.

	Counterpoint repair	SQL repair
Artifact	MusicXML upper voice with fixed cantus firmus	SQL query paired with schema and reference
Independent target	Weighted satisfaction of relational counterpoint rules	Result-set agreement over generated databases
Aligned feedback	Graded rule satisfaction on search instances	Graded execution agreement
Proxy feedback	Eight surface musical statistics	Seven non-executing style features
Distinct failure	Valid music with accumulated rule violations	Tidy-looking query with incorrect or failed execution

Table 2: Endpoint results after six generations. ΔV_{evol} measures visible-score ascent and E_{target} the held-out target effect relative to B0; "Withheld" omits score and error reports from the proposal prompt.

Run	Selection feedback	ΔV_{evol}	E_{target}	95% CI for E_{target}	p
Primary study: counterpoint repair					
Primary	Graded rules	$+0.034$	$+0.032$	$[+0.010, +0.059]$	0.250
Primary	Symbolic proxy	$+0.098$	-0.292	$[-0.346, -0.237]$	0.007
Withheld	Graded rules	$+0.042$	$+0.036$	$[+0.013, +0.064]$	0.250
Withheld	Symbolic proxy	$+0.063$	-0.171	$[-0.246, -0.092]$	0.037
Directional cross-domain check: SQL repair					
SQL	Execution	$+0.117$	$+0.117$	$[+0.062, +0.179]$	3.1×10^{-4}
SQL	Style proxy	$+0.222$	-0.119	$[-0.177, -0.064]$	2.9×10^{-4}

Counterpoint p-values are Holm-adjusted; confidence intervals and SQL p-values are unadjusted.

4.5 Implementation and statistical protocol

All evolution conditions run for six generations with up to two proposed scaffold revisions per generation. Executing one scaffold on one instance is limited to five base-model calls and 24,000 generated tokens; the scaffold decides how many calls to use within that cap. The base model is `deepseek-flash`. Candidate programs must pass static syntax checks and execute in a sandbox. Music inference uses 16 held-out instances. For each instance, effects are averaged across seeds before pairing, so the Wilcoxon test and instance bootstrap use $n = 16$, not 16 times the number of seeds; the primary graded-rule and symbolic-proxy rows use six seeds, and the withheld rerun uses two. SQL analogously averages each instance across three seeds before its main paired test ($n = 64$); task- and template-level checks aggregate those instances to $n = 48$ and $n = 12$. We report paired Wilcoxon signed-rank tests (Wilcoxon, 1945), unadjusted paired-bootstrap confidence intervals, and Holm-corrected music p-values (Holm, 1979). The music correction family contains 15 prespecified comparisons: three prompt-only conditions and all six evaluator conditions in each of the primary and withheld code-evolution arms; Appendix A lists them. For SQL, the style-proxy sign persists at all three aggregation levels and is detected at the instance and task levels, but not across the 12 templates. Complete run configurations and scorer implementations are provided in the supplementary material.

5 Results

5.1 Visible and target scores reverse under proxy feedback

Visible feedback rises in every row of Table 2. Target effects relative to B0 are positive in the three aligned rows and negative in the three proxy rows. The two counterpoint aligned effects do not survive family-wise correction, whereas the SQL aligned effect is detected in its prespecified paired test. Within each paired comparison, the selection evaluator is the experimental intervention.

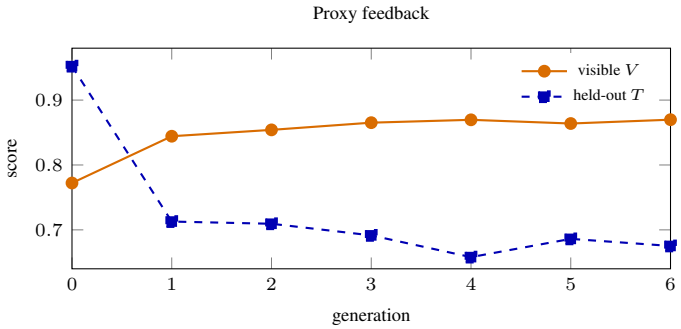


Figure 2: Counterpoint trajectory under proxy feedback. The trajectory is descriptive; we omit the aligned trajectory because visible and target scores use the same graded-rule function in that condition.

Calibration establishes that the feedback functions provide qualitatively different orderings before any scaffold is evolved. Graded rule feedback ranks every clean–violated pair correctly in the calibration set. The symbolic proxy separates the same pairs by only 0.004 on average and ranks the clean artifact higher in 57.5% of cases. Local search can nevertheless increase it substantially while decreasing the target. The proxy therefore supplies a clear optimization direction without reliably preserving the target ordering, explaining how the loop can make measurable progress while selecting worse repairs.

In the primary counterpoint rows, the visible graded-rule score rises by 0.034 from the start to the final scaffold. Relative to B0, that final scaffold has a target effect of $+0.032$. Under proxy feedback, the corresponding quantities are $+0.098$ and -0.292. The target loss remains significant after Holm correction ($p = 0.007$; unadjusted 95% CI $[-0.346, -0.237]$) and is 3.68 times the minimum detectable effect. The aligned estimate is positive but does not survive correction ($p_{\text{Holm}} = 0.250$), so it serves as a directional control rather than a detected gain. Figure 2 shows the proxy–target divergence over generations; endpoint estimates and intervals are reported in Table 2.

The independent rerun that withholds the score and error report from the proposal prompt reproduces both signs: graded-rule feedback changes the target by $+0.036$, while proxy feedback changes it by -0.171 ($p_{\text{Holm}} = 0.037$).

Across the preregistered split and three post-hoc variants, graded-rule target effects range from $+0.032$ to $+0.187$, while proxy-conditioned effects range from -0.135 to -0.366; Appendix A reports all four. The rerun shows that proxy-conditioned degradation does not require showing the proposal model the score or error report: outer selection still favors lower-target scaffolds.

5.2 Scaffold evolution exceeds fixed-solver selection

Proxy-based best-of-five selection can itself reduce the target score. The fixed-solver control generates exactly five independent repairs per instance and selects the highest-scoring repair, repeated four times. Against B0 on the primary split, its target effect is -0.070, while the scaffold-evolution effect against B0 is -0.292. Their descriptive mean-denominator ratio is 4.17, and the evolved proxy scaffold averages 4.47 calls per instance. On the higher-headroom F4 variant, four positive comparisons give evolution-to-control ratios of 1.85–2.10; the graded-rule condition itself is $1.86\times$. These point estimates indicate that the effect observed under fixed-solver selection is smaller than that observed under the full evolution procedure.

All four fixed-solver repetitions lose target score. Against the same B0 baseline, the mean effects are -0.070 for fixed-solver selection and -0.292 for evolution. Their magnitude ratio is the descriptive estimate $\Gamma = 4.17$, with repeat-specific ratios from 3.25 to 5.75; no confidence interval or hypothesis test is attached to this ratio. Final graded-rule and proxy scaffolds average 3.97 and 4.47 calls per instance while moving the target in opposite directions. Because the fixed control uses all five evaluation-time calls, the larger evolved effect is not explained by a larger per-instance infer-

Table 3: Primary counterpoint attribution under a common five-call cap. Both effects use the four-repeat single-shot B0 baseline; they are not a direct fixed-to-evolved contrast. The resulting Γ is a descriptive point estimate. Calls are evaluation-time base-model calls per instance.

System	Scaffold rewritten?	Mean calls/instance	Target effect vs. B0
Single-shot fixed baseline (B0)	No	1.00	0.000 (definition)
Best-of-five fixed solver	No	5.00	-0.070
Evolving scaffold	Yes	4.47	-0.292

ence budget. The comparison concerns the full evolution procedure and does not isolate program rewriting from the compute used to search for programs.

5.3 SQL REPAIR SHOWS THE SAME SIGN REVERSAL

With execution feedback, visible ascent and the target effect are both $+0.117$ ($p = 3.1 \times 10^{-4}$), and all three seeds improve. With style feedback, visible ascent is $+0.222$ while the target effect is -0.119 ($p = 2.9 \times 10^{-4}$); all three seed-level effects are negative. These p-values use the 64-instance analysis after averaging across seeds. At the task level ($n = 48$), the effects remain detected and retain their signs. At the template level ($n = 12$), both signs persist, but the style-proxy result is not detected ($\Delta/\mathrm{MDE} = -0.865$, $p = 0.388$). SQL therefore provides a directional cross-domain check with greater template-level uncertainty than the counterpoint result.

Execution feedback has zero regret by construction on the common re-scoring set. Under the style proxy, pooled regret is $+0.129$ and within-generation regret is $+0.101$, with positive pooled regret in every seed. In one seed, the starting scaffold has no false-good queries—proxy at least 0.8 but target score zero—whereas the proxy-best candidate in the full evolved pool has $17/64$; the proxy-lowest candidate has $53/64$. The first generation accounts for 86% of the counterpoint trajectory loss and 111% of the SQL loss, after which counterpoint continues to deteriorate while SQL partially rebounds.

5.4 RETAINED SCAFFOLDS ENCODE EVALUATOR-SPECIFIC PROCEDURES

The source audit documents persistent, feedback-specific code changes. Among 20 archived counterpoint scaffolds, 18 differ from the 83-line initial scaffold, 10 introduce an internal candidate-selection rule, 11 rewrite the generation prompt, six can return the input as a fallback, and 16 add an empty-response fallback. In exactly three symbolic-proxy scaffolds, prompts name exposed components such as consonance rate, interval variety, motion balance, pitch variety, step ratio, key fit, leap softness, and range fit.

All six audited SQL scaffolds call their available scorer. Every execution-feedback scaffold also calls the execution report, whereas only one of three style-feedback scaffolds does so; the latter implement string-level repair without an execution check. These structural counts do not establish that a particular edit caused an outcome, but they show how evaluator-specific procedures persist after selection. Appendix A gives the audit counts.

6 CONCLUSION

We studied whether a rising optimization score in scaffold-level self-improvement is evidence that an independent target has improved. By separating visible feedback V from a held-out target T and changing only V across paired conditions, we found that scaffold evolution can raise its selection score while lowering the independently measured target, and that retained scaffolds encode evaluator-specific prompting, checking, and selection rules. Optimizing against a partial evaluator is therefore insufficient evidence of self-improvement; such claims should be accompanied by independent held-out target evaluation.

Reproducibility Statement

Section 4 specifies the two domains, feedback functions, data partitions, resource limits, and statistical protocol. Appendix A summarizes validity checks, proxy calibration, robustness results, seed-level diagnostics, and program-audit counts. The supplementary material will include complete run configurations, the initial and retained scaffolds, per-instance visible and target scores, scorer implementations, data-generation scripts, random seeds, and scripts required to reproduce the reported tables and figures. Held-out instances and their target diagnostics remain inaccessible to the improvement loop during reproduction.

AI Use Statement

Generative AI tools were used to assist with literature discovery, language editing, manuscript organization, LaTeX preparation, and code development. All cited works were read and verified by the authors. All research questions, experimental designs, implementations, data, analyses, numerical results, and scientific claims were produced or verified by the authors. The authors take full responsibility for the content of the paper.

References

Torsten Anders and Eduardo Reck Miranda. Constraint programming systems for modeling music theories and composition. *ACM Computing Surveys*, 43(4):1–38, 2011.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Meilin Chen, Jian Tian, Liang Ma, Di Xie, Weijie Chen, and Jiang Zhu. Unbiased evaluation of large language models from a causal perspective. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 9105–9120, 2025.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2024.

Johann Joseph Fux. *Gradus ad Parnassum*. Johann Peter van Ghelen, Vienna, 1725.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.

Michael Good. MusicXML: An internet-friendly format for sheet music. XML Conference and Exposition, 2001.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujia Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, 2024.

Gaëtan Hadjeres, François Pachet, and Frank Nielsen. DeepBach: A steerable model for Bach chorales generation. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1362–1371, 2017.

Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024a.

Yujia Huang, Adishree Ghatare, Yuanzhe Liu, Ziniu Hu, Qinsheng Zhang, Chandramouli Shama Sastry, Siddharth Gururani, Sageev Oore, and Yisong Yue. Symbolic music generation with non-differentiable rule guided diffusion. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 19772–19797, 2024b.

Natasha Jaques, Shixiang Gu, Richard E. Turner, and Douglas Eck. Generating music by fine-tuning recurrent neural networks with reinforcement learning. In *Deep Reinforcement Learning Workshop, Advances in Neural Information Processing Systems*, 2016.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A. Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. AI safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

David Manheim and Scott Garrabrant. Categorizing variants of goodhart's law. *arXiv preprint arXiv:1803.04585*, 2018.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. In *ICLR Workshop on Scaling Self-Improving Foundation Models without Human Supervision*, 2025.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

Jürgen Schmidhuber. Gödel machines: Self-referential universal problem solving machines. *arXiv preprint cs/0309048*, 2003.

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*, volume 36, 2023.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, 2018.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304*, 2023.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-SQL with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 396–411, 2020.

Tingyu Zhu, Haoyu Liu, Ziyu Wang, Zhimin Jiang, and Zeyu Zheng. Efficient fine-grained guidance for diffusion model based symbolic music generation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 79938–79962, 2025.

A Additional Experimental Details

Statistical protocol. Confirmatory counterpoint comparisons use paired Wilcoxon signed-rank tests, unadjusted paired-bootstrap confidence intervals, and Holm correction over a prespecified family of 15 comparisons on the primary F1 split. The family comprises three prompt-only B1 conditions (binary rules, graded rules, and the symbolic proxy), six primary code-evolution B3 conditions, and six withheld-information B4 conditions. The six evaluators in B3 and B4 are binary rules, graded rules, noisy graded rules, the symbolic proxy, a ridge proxy, and a frozen language-model judge. Only the four graded-rule and symbolic-proxy B3/B4 rows are shown in Table 2; Holm correction uses all 15 tests. For each music instance, effects are averaged over seeds before pairing, giving $n = 16$ for every reported test despite differing seed counts. The reported effect-to-MDE ratio compares the target effect with the minimum detectable paired effect at 80% power. SQL similarly averages over three seeds before the instance-level test ($n = 64$) and additionally reports task-level ($n = 48$) and template-level ($n = 12$) aggregation.

Music validity. Candidate MusicXML files must parse, preserve the cantus firmus and rhythmic grid, and satisfy the interface constraints before scoring. Invalid outputs receive the specified floor behavior and cannot gain access to hidden evaluation diagnostics. The target aggregates rule satisfaction only after these structural checks.

SQL validity. Each candidate and reference query is evaluated on the same generated in-memory database for that instance. Execution feedback uses graded result-set agreement. The style proxy uses only static properties and never observes execution outputs.

B Interpreting the Scope of Self-Improvement

The evolving object is a persistent executable scaffold that can alter prompts, allocate calls, request candidates, invoke the visible scorer, and implement selection and fallback rules. The base language model and outer mutation-and-retention algorithm remain unchanged. The results therefore concern scaffold-level procedural adaptation, not weight updates or unrestricted recursive self-improvement.

Table 4: Pre-evolution calibration. Neither optimizer uses a language model; both expose proxy ascent directions that reduce independent rule satisfaction.

Optimization target	Final proxy	Target: start → end	Search
Symbolic surface proxy	0.910	0.856 → 0.605	400 mutations
Learned ridge proxy	0.992	0.856 → 0.424	400 mutations

Table 5: Counterpoint robustness variants. The primary held-out split was preregistered; the other three variants were selected after the primary run and are exploratory. The target-change sign is invariant across all four.

Evaluation variant	Rule-feedback E_{target}	Proxy-feedback E_{target}
Primary held-out split	$+0.032$	-0.292
Surface-form variant	$+0.042$	-0.366
Difficulty-restricted variant	$+0.042$	-0.317
Information-restricted variant	$+0.187$	-0.135

Table 6: SQL diagnostics. A false-good output receives proxy score at least 0.8 but zero execution score.

Diagnostic	Execution feedback	Style proxy
Seed directions	$3/3$ positive	$3/3$ negative
Target effect by seed	all positive	$-0.260, -0.083, -0.015$
Selection regret, pooled	0.000	$+0.129$
False-good outputs, proxy-best candidate	$0/64$	$17/64$
False-good outputs, proxy-lowest candidate	$0/64$	$53/64$
Final calls per instance	1.67–2.06	1.00

Table 7: Structural audit of retained programs. Counts document persistent, feedback-specific code changes but do not attribute an outcome to any individual edit.

Observed program change	Scaffolds
Music scaffold differs from the 83-line seed	$18/20$
Music scaffold adds an internal selection rule	$10/20$
Music scaffold rewrites its generation prompt	$11/20$
Music scaffold can return the input as fallback	$6/20$
Music scaffold adds empty-response fallback	$16/20$
Proxy scaffold names exposed proxy components	3
SQL scaffold calls the available scorer	$6/6$
Execution-feedback SQL scaffold calls execution report	$3/3$
Style-feedback SQL scaffold calls execution report	$1/3$