WHEN COUNTEREXAMPLES DON'T TEACH: LEARNER-SPECIFIC FEEDBACK IN LLM REPAIR

Anonymous authors
Paper under double-blind review

ABSTRACT

Counterexamples are widely used to guide program synthesis and repair: a verifier exposes a concrete failure, and the learner revises its current solution from that feedback. Existing selection strategies largely judge counterexamples by how informative they are about the hypothesis space. This is natural for symbolic learners, but less obvious when the updater is an LLM, whose response to the same valid feedback can vary substantially. We show that formal informativeness is often a poor predictor of whether an LLM will make a useful repair. The mismatch appears across several LLMs and executable task families. Motivated by this observation, we introduce TEACHCE, which selects counterexamples according to the repair consequences they are expected to induce in the current learner, and an adaptive verifier-guided variant that reuses repair attempts when needed. Across controlled repair tasks and sequential settings, this learner-specific view improves the quality–compute trade-off over symbolic selection. On public student-program benchmarks, we further show that combining consequence-aware selection with adaptive verifier-guided rollout reuse improves repair efficiency under a limited inference budget. These results suggest that, once LLMs enter the repair loop, choosing feedback requires modeling not only what the evidence says, but also how the learner will respond to it.

1 INTRODUCTION

Program synthesis and repair increasingly combine two complementary capabilities. A verifier can identify concrete failures with precise executable feedback, while a large language model (LLM) can use that feedback to propose flexible revisions. Counterexample-guided inductive synthesis (CEGIS) (Solar-Lezama et al., 2006) provides a canonical instance of this interaction: a learner proposes a program, a verifier returns an input on which it disagrees with the specification, and the learner revises its hypothesis from that labeled failure. Related ideas also appear in query learning and machine teaching (Angluin, 1987; Zhu, 2015). The quality of the feedback is therefore central to how efficiently the learner can improve.

When several counterexamples are available, classical selection rules usually rank them by how informative they are about the hypothesis space. A teacher may prefer a short witness (Jha & Seshia, 2014), a balanced version-space split (Angluin, 1987) or an example with high entropy reduction (MacKay, 1992). These criteria are well motivated for symbolic learners, where a valid counterexample acts as a hard constraint: once shown, every inconsistent hypothesis can be removed. Under this model, a more informative example is also a plausible better teaching action.

This connection becomes less clear when the updater is an LLM. A verifier may still provide a perfectly valid counterexample, but the model need not incorporate it in the intended way. The same feedback can lead to a structural repair (Jimenez et al., 2024), a patch that only fixes the displayed case, no useful change, or a regression on behavior that was already correct. Existing techniques such as reflection (Madaan et al., 2023), repeated repair (Shinn et al., 2023), and richer counterexample feedback (Liu et al., 2026) can improve this process, but they do not remove the dependence of the update on the particular learner and its current state.

This observation motivates the distinction at the center of this work. *Formal informativeness* measures how strongly a counterexample constrains the hypothesis space, whereas *teachability* concerns the semantic change that the counterexample actually induces in a particular learner. A counterexample

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.

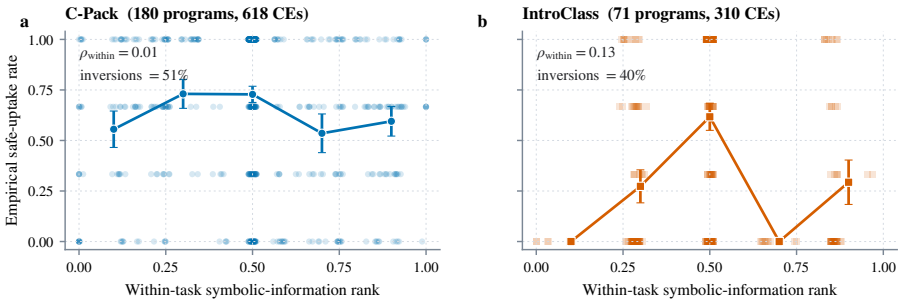


Figure 1: **Formal rank does not reliably predict safe repair on real programs.** Counterexamples are averaged over three DeepSeek repairs. Within-task correlations are weak, and many C-Pack and IntroClass candidate pairs reverse their formal and empirical teaching order.

can be formally informative yet difficult for an LLM to use, while a weaker counterexample can trigger a more reliable repair.

Figure 1 shows that this mismatch is already visible on real student programs. Across C-Pack (Orvalho et al., 2024) and IntroClass (Le Goues et al., 2015), formally stronger counterexamples are only weakly associated with safer repairs, and many candidate pairs reverse their formal and empirical order. We further study this phenomenon in controlled executable environments, where repair consequences can be measured exactly. Across Datalog (Cropper & Morel, 2021), Regular Expressions (Gulwani, 2011), and a list-program DSL, formal counterexample rank remains weakly related to semantic progress. These failures are not random: repair success depends strongly on the required semantic transition, and a counterexample that is easy to fix locally may still trigger regressions elsewhere. This suggests that effective counterexample selection should model not only the information contained in feedback, but also the learner transition it induces.

Based on this view, we introduce TEACHCE, a learner-specific counterexample teacher that predicts the repair consequences induced by each candidate and uses these predictions to rank the available feedback. We further develop TEACHCE-AR, an adaptive variant that integrates selection with verifier-guided rollout reuse. Each generated repair is executed and retained as a possible final solution. A second counterexample is requested only when the first repair is not sufficiently supported by the verifier. This turns counterexample selection into a compute-aware decision about both which feedback to show and when additional feedback is worth the cost. Extensive experiments show that this learner-specific formulation improves the quality–compute trade-off across controlled repair tasks, several LLMs, sequential CEGIS, and public C benchmarks. Our contributions are as follows:

(1) **We identify a learner-side failure of symbolic counterexample selection.** We show that formal informativeness and LLM teachability can rank the same counterexamples very differently, both in controlled executable tasks and on real student programs.

(2) **We formulate counterexample selection as learner-specific consequence prediction.** We propose TEACHCE, which selects feedback according to the repair it is expected to induce, and TEACHCE-AR, which combines this ranking with adaptive verifier-guided rollout reuse.

(3) **We connect the empirical phenomenon to decision quality.** We provide one-step and finite-horizon analyses of learner-specific teaching and evaluate the resulting policies across multiple task families, several LLMs, sequential repair, and public C benchmarks.

2 RELATED WORK

Counterexamples in learning and synthesis. Counterexamples provide a classical interface between verification and learning. In equivalence-query learning, a learner proposes a hypothesis and receives a counterexample whenever it disagrees with the target (Angluin, 1987). CEGIS adopts the same interaction for program synthesis, repeatedly using verifier-generated failures to refine the current candidate (Solar-Lezama et al., 2006; Abate et al., 2018). Related formulations in inductive logic programming and symbolic concept learning likewise treat labeled examples as constraints on a

structured hypothesis space (Cropper & Dumančić, 2022). These settings motivate selection criteria based on version-space reduction, balanced splitting, or information gain: a useful counterexample is one that rules out many competing hypotheses. Such criteria are well matched to consistent symbolic learners, for which satisfying the new constraint is part of the update rule.

Machine teaching and learner-aware selection. Machine teaching studies how a teacher should choose training examples for a particular learner rather than passively sampling data (Zhu, 2015; Zhu et al., 2018). Learning-to-teach methods further learn example-selection policies from the response of a student model (Fan et al., 2018), while learner-aware and preference-based teaching explicitly account for learner-specific biases, preferences, or constraints (Tschiatschek et al., 2019; Mansouri et al., 2019). This work establishes that the same teaching example need not have the same value for different learners.

Feedback and self-correction in LLMs. A growing literature studies how LLMs can refine their outputs using verbal, self-generated, or externally verified feedback (Lalonde et al., 2026). Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) iterate between generation and critique, while CRITIC (Gou et al., 2024) uses external tools to obtain feedback that can be checked independently of the model. Broader studies of LLM correction similarly distinguish self-feedback from reliable external feedback and show that the source and quality of feedback strongly affect whether refinement succeeds (Pan et al., 2024; Kamoi et al., 2024; Stechly et al., 2025). These works primarily study how feedback is generated, represented, or accumulated.

LLM-based program repair and verifier-guided refinement. LLMs have increasingly been combined with execution and test feedback for automated program repair (Yang et al., 2024; Orvalho et al., 2025a). Early work showed that code LLMs can serve as flexible multilingual repair engines (Joshi et al., 2023), while conversational approaches such as ChatRepair (Xia & Zhang, 2024) iteratively return compilation and test failures to the model. RePair (Zhao et al., 2024) incorporates process-level feedback into multi-step repair, and more recent methods improve refinement by designing richer or more diagnostic feedback, including contrastive test pairs (Kong et al., 2025). Closest to our setting, Orvalho et al. (Orvalho et al., 2025b) place an LLM inside a CEGIS loop and feed test-suite counterexamples back to the model, while Liu et al. (2026) study counterexample clusters, regularization, and reflection for LLM-based regular-expression induction.

3 From Informativeness to Teachability

At a CEGIS step, the verifier may expose multiple valid failures of the current hypothesis, making counterexample selection a decision over which feedback to provide. Classical strategies rank candidates by their informativeness about the hypothesis space, which is appropriate for consistent symbolic learners where counterexamples eliminate inconsistent hypotheses. An LLM updater differs: a counterexample only initiates a stochastic repair process, and the most informative evidence need not induce the most effective revision. We formalize this distinction below.

Symbolic informativeness. At step t, let $h_t \in \mathcal{H}$ denote the current hypothesis and $h^\star$ the target behavior. The verifier returns a candidate pool $\mathcal{C}_t$ containing inputs on which h_t disagrees with $h^\star$. Let $V_t \subseteq \mathcal{H}$ be the version space consistent with the evidence observed so far. In the binary setting, a standard balanced-split criterion is

$$E_t(x) = \min\Big(|\{h \in V_t : h(x) = 0\}|, |\{h \in V_t : h(x) = 1\}|\Big), \qquad x \in \mathcal{C}_t. \tag{1}$$

Entropy gain and label-conditioned elimination provide closely related alternatives. For a consistent symbolic learner, $E_t(x)$ has a direct operational meaning: after the label of x is revealed, hypotheses that disagree with it are removed from V_t. Ranking counterexamples by formal informativeness is therefore also a reasonable way to rank their teaching value.

LLM repair changes the decision problem. For an LLM, observing a counterexample does not determine the next hypothesis. Under a fixed feedback format f, we model the update as

$$h_{t+1} \sim q_\theta(\cdot \mid h_t, x, h_t(x), h^\star(x), f)\,, \tag{2}$$

where q_θ is the repair distribution induced by the learner. The formal score $E_t(x)$ is known before the model is called, whereas the quality of the resulting repair depends on this learner-specific transition. Counterexample selection has therefore changed from ranking evidence to choosing an intervention.

Conceptually, the relevant quantity is the expected consequence of that intervention. Let $u_t(h')$ denote the utility of a possible repaired hypothesis h' at state t. We define the *teachability* of x for the current learner as

$$T_t(x) = \mathbb{E}_{h' \sim q_\theta(\cdot \mid h_t, x, h_t(x), h^\star(x), f)} \big[u_t(h') \big]. \tag{3}$$

Unlike $E_t(x)$, $T_t(x)$ depends jointly on the counterexample, the current hypothesis, and the learner. Consequently, a formally stronger counterexample can have lower teaching value if it tends to trigger a poor repair, while a weaker one can be preferable if the learner responds to it reliably. The rest of the paper studies how to measure and predict this learner-dependent value.

What counts as a useful repair? A repair should do more than fix the single input that triggered it. We call the update *local* if h_{t+1} corrects the displayed counterexample x. To test whether the inferred change extends beyond that example, let $\mathcal{N}(x)$ denote a precomputed neighborhood of failures that share the same underlying repair signature. ***Structural uptake*** requires h_{t+1} to correct every example in $\mathcal{N}(x)$. A useful repair should also preserve behavior that was already correct. Let R_t be a regression set on which h_t and $h^\star$ agree. We define *safe uptake* as

$$U_{\text{safe}} = \prod_{z \in \mathcal{N}(x)} \mathbf{1}[h_{t+1}(z) = h^\star(z)] \prod_{r \in R_t} \mathbf{1}[h_{t+1}(r) = h^\star(r)]. \tag{4}$$

This criterion separates genuine structural repair from a local patch that fixes the observed failure while introducing regressions elsewhere.

We measure the net semantic change induced by the repair. For a finite executable universe $\mathcal{X}$, let $\mathrm{Err}_{\mathcal{X}}(h, h^\star)$ be the number of inputs on which h disagrees with the target. The realized progress is

$$G_t = \mathrm{Err}_{\mathcal{X}}(h_t, h^\star) - \mathrm{Err}_{\mathcal{X}}(h_{t+1}, h^\star), \qquad \bar{G}_t = \frac{G_t}{\mathrm{Err}_{\mathcal{X}}(h_t, h^\star)}. \tag{5}$$

Positive G_t means that the repair reduces total semantic error. Negative G_t means that it makes the hypothesis worse overall. This distinction is important because local success and global progress need not agree.

4 Learner-Specific Counterexample Teaching

Ideally, a teacher would rank each $x \in \mathcal{C}_t$ by its teachability $T_t(x)$. Directly estimating this quantity by repeatedly trying every candidate would itself require many learner calls (Setlur et al., 2025). TeachCE instead predicts the consequences of each candidate before querying the learner, and TeachCE-AR combines this ranking with verifier-guided reuse of the resulting repairs. An overview of our method is shown in Figure 2.

Consequence-aware selection. Let $\mathcal{F}_t$ contain the information available before the learner call, including the current hypothesis, verifier-provided evidence, formal scores, and pool-level statistics. For each $x \in \mathcal{C}_t$, TeachCE predicts several consequences of the induced repair:

$$Z(x) = \big(Z_{\text{exact}},\ \bar{G},\ \widetilde{G},\ Z_{\text{safe}}, Z_{\text{reg}},\ Z_{\text{cat}},\ D\big), \qquad \widehat{m}_j(x) \approx \mathbb{E}[Z_j(x) \mid \mathcal{F}_t], \tag{6}$$

where Z_{exact} denotes exact recovery, Z_{safe} safe uptake, and Z_{reg} regression. In practice, we use a six-head consequence predictor by selecting the normalized progress $\bar{G}$. To separate ordinary progress from severe failures, we use

$$\widetilde{G} = \mathrm{clip}(\bar{G}, -1, 1), \qquad Z_{\text{cat}} = \mathbf{1}[\bar{G} < -1], \qquad D = \log(1 + \max\{0, -\bar{G}\}).$$

The predictions are combined into

$$\widehat{A}_w(x) = \sum_{j \in \mathcal{J}} w_j \widehat{m}_j(x) + w_I \widetilde{E}_t(x), \tag{7}$$

where $\widetilde{E}_t(x)$ is the normalized formal-information score. Thus, the acquisition can trade off expected repair benefit, regression risk, and formal information within a single decision rule.

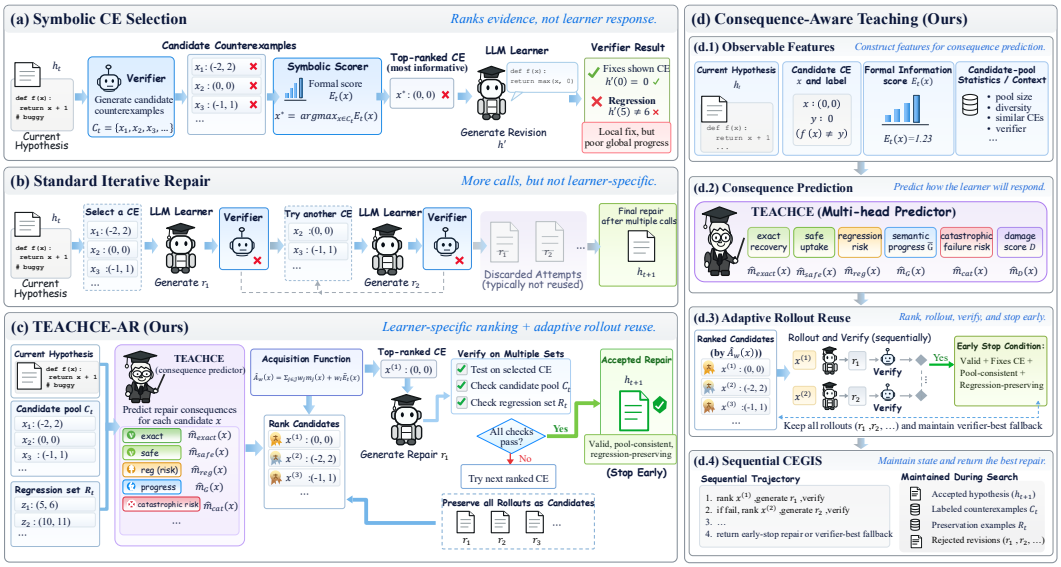


Figure 2: Overview of our proposed TEACHCE and TEACHCE-AR.

Verifier-guided adaptive reuse. A predicted ranking cannot remove the stochasticity of the actual LLM update. TEACHCE-AR therefore queries candidates in decreasing order of $\widehat{A}_w(x)$ and immediately executes each returned repair against the labeled counterexample pool and regression suite. Pool consistency means that the generated repair satisfies all currently maintained verifier constraints in the counterexample pool C_t. A repair that satisfies the available evidence and preserves previously correct behavior is returned. Otherwise a different counterexample is tried and the verifier-best repair observed so far is retained.

The key design is that every rollout is itself a candidate final repair, so additional compute is spent only when the observed transition does not justify stopping. The primary setting uses at most two distinct counterexamples. The pseudocode are shown in Appendix A.

We next provide a decision-theoretic characterization of learner-specific teaching. The finite-horizon analysis considers an ideal consequence-aware policy that optimizes estimated future utility, which motivates our practical sequential router.

Proposition 1 (Failure under a monotone logistic updater)**.** *For every $\epsilon \in (0, 1)$, there is a finite two-counterexample problem in which the unique symbolic-score maximizer attains less than an ϵ fraction of the best expected one-step progress, even when the learner has only two possible next states (the current and target hypotheses) and its uptake probability is logistic and monotone in a scalar difficulty.*

The construction separates the amount of hypothesis-space elimination from the probability that the learner successfully performs the corresponding repair. A formally stronger counterexample can therefore have arbitrarily lower teaching value. The construction and proof are in Appendix B.

Conversely, let $m_j(x) = \mathbb{E}[Z_j(x) \mid \mathcal{F}_t]$ be the true conditional consequence, and let $A^\star_w$ denote Equation 7 with $\widehat{m}_j$ replaced by m_j.

Theorem 2 (One-step optimality and plug-in regret)**.** *Any maximizer of $A^\star_w$ maximizes conditional expected utility over the finite candidate set. Moreover, if $|\widehat{m}_j(x) - m_j(x)| \le \eta_j$ uniformly and $\widehat{x}$ maximizes the plug-in acquisition, then*

$$A^\star_w(x^\star) - A^\star_w(\widehat{x}) \le 2 \sum_{j \in \mathcal{J}} |w_j| \eta_j. \tag{8}$$

Hence TEACHCE can be viewed as a plug-in approximation to the optimal one-step learner-specific teacher: better consequence prediction translates directly into lower teaching regret. The same

Table 1: **Controlled uptake audit.** Local correction substantially overestimates successful teaching. TRIR denotes the fraction of comparable candidate pairs whose symbolic and empirical teaching orders disagree.

Family	Repairs	Local	Structural	Safe	Exact	TRIR
Datalog	6,000	.831	.460	.327	.236	.490
Regex	5,760	.857	.599	.364	.150	.543
List DSL	5,760	.602	.433	.304	.271	.468

objective can incorporate model cost by subtracting $\kappa c(a)$ for a repair plan a, giving the quality–compute interpretation of TEACHCE-AR. Proofs and the cost-aware formulation are given in Appendix B.

Sequential extension. For multi-round CEGIS, the controller additionally maintains the accepted hypothesis, accumulated verifier evidence, rejected revisions, and a regression suite. The current disagreement pool is summarized by whether the required repairs are generalizing $(0 \to 1)$, specializing $(1 \to 0)$, or mixed. A frozen router uses this observable state to select the next counterexample pair, and a proposed hypothesis is committed only if it remains consistent with accumulated evidence and preserves the regression suite. Full routing and memory rules are given in Appendix A.

The one-step decision view also extends to this finite-horizon setting. If $\widehat{Q}_t$ approximates the optimal action value $Q_t^\star$ with uniform error η_t, a greedy policy satisfies

$$V_1^\star(s_1) - V_1^{\widehat{\pi}}(s_1) \le 2 \sum_{t=1}^{T} \eta_t. \tag{9}$$

The complete finite-horizon theorem, assumptions, and proof are provided in Appendix B.

5 EXPERIMENTS

We organize the experiments around four questions: **RQ1:** Does formal informativeness predict the repair induced by an LLM? **RQ2:** Does learner-specific counterexample selection improve repair under a matched compute budget? **RQ3:** Does the same principle help in sequential CEGIS? **RQ4:** Does the resulting policy transfer across learner models and to real program repair? Unless stated otherwise, uncertainty is estimated by resampling tasks rather than individual model calls. Full task construction, prompts, model settings, and evaluation protocols are provided in Appendix A.4.

5.1 RQ1: DOES FORMAL INFORMATIVENESS PREDICT REALIZED TEACHING?

We first test the assumption underlying classical counterexample selection: whether formal informativeness predicts useful LLM repair. The controlled audit contains 5,840 counterexamples across Datalog, Regex, and List DSL, with three independent repairs per candidate.

Table 1 shows that local correction is a poor proxy for learning. In Datalog, although 83.1% of repairs fix the displayed counterexample, only 32.7% achieve safe uptake and 23.6% recover the target exactly. Similar gaps appear in Regex and List DSL, indicating that LLMs may satisfy observed failures without recovering the underlying rule.

More importantly, symbolic ranking poorly predicts semantic improvement. The Spearman correlation between maximin version-space score and realized progress is only 0.016, -0.078, and 0.071 across the three domains, with 46.8–54.3% of candidate pairs reversing their symbolic and empirical orders. Alternative criteria, including entropy gain and label-conditioned elimination, exhibit the same mismatch (Appendix D).

This finding is not explained by sampling noise. On a high-replication subset, increasing repairs per counterexample from three to twenty improves ranking stability, but the noise-corrected correlation remains 0.007, with 50.4% of confidently ordered pairs still reversed (Figure 3). Model changes affect overall repair success but do not restore the relationship between symbolic rank and semantic

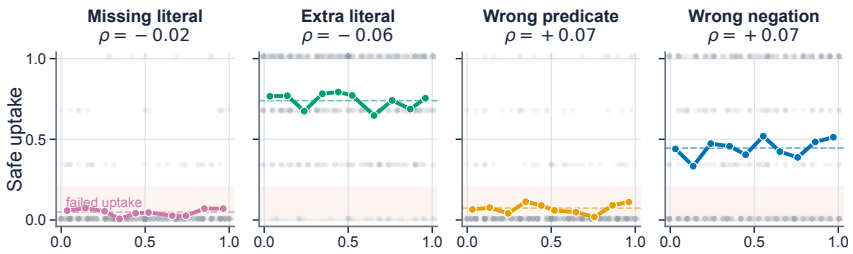


Figure 3: **Formal rank is a poor predictor of safe uptake.** Each point is a counterexample averaged over three repairs. Traces show rank-binned means within the required repair class. Variation across repair classes is substantially larger than any trend with symbolic rank.

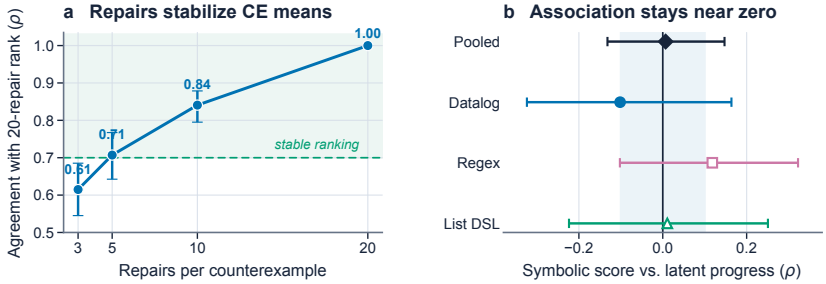


Figure 4: **Increasing replication stabilizes empirical teaching ranks without recovering a symbolic relationship.** (a) Agreement with the 20-repair counterexample ranking increases as more repairs are observed. (b) After accounting for sampling noise, the association between symbolic score and latent repair progress remains close to zero across all three task families. Whiskers denote 95% resampling intervals.

progress. Mixed-effect analyses further show that repair outcomes are better explained by the required learner transition than by counterexample informativeness (Figure 5).

These results answer RQ1 negatively. Formal informativeness captures how strongly a counterexample constrains the hypothesis space, but it does not reliably indicate which counterexample will induce the better repair from an LLM.

5.2 RQ2: Does Learner-Specific Selection Improve Repair?

We next test whether modeling the learner response leads to better counterexample choices under a matched compute budget. We use two controlled comparisons. The first evaluates different ways of spending up to two learner calls, while the second fixes the adaptive executor and changes only the counterexample ordering.

Table 2(a) compares TeachCE-AR with repetition, reflection, and distinct-counterexample reuse on 240 disjoint tasks. All methods draw from the same stored rollout matrix and use the same verifier. TeachCE-AR achieves 35.8% exact recovery and 41.1% bounded progress with 1.75 calls on average. The strongest alternative reaches 31.7% exact recovery, while catastrophic regressions are also less frequent under TeachCE-AR. The gain therefore cannot be explained simply by allowing a second sample or by adding verifier feedback after generation.

Table 2(b) isolates the effect of counterexample ordering on 120 fresh tasks. Every method uses the same adaptive two-call executor, so the only difference is which candidate is tried first and, when necessary, next. Symbolic information performs poorly in this setting: maximum-information selection reaches 29.2% exact recovery, compared with 39.2% for TeachCE-AR. The learner-specific policy also achieves the highest safe uptake at 48.3%, while using fewer calls and tokens than the other displayed orderings. Predicting exact recovery or progress alone improves over symbolic selection, but remains below the joint consequence model.

Table 2: **Matched adaptive repair comparisons.** (a) Repair reuse under a shared rollout budget ($n = 240$). (b) Counterexample ordering with a shared cap-two verifier on disjoint tasks ($n = 120$). Outcomes are percentages. ↓ marks lower-is-better.

(a) Repair reuse, $n = 240$

Policy	Exact	BProg	Cat↓	Calls↓
Shortest, repeat-2	25.0	15.4	12.5	2.00
Max symbolic, repeat-2	25.4	17.5	12.1	2.00
TEACHCE same-CE, repeat-2	31.7	35.4	4.2	2.00
TEACHCE reflection-2	31.7	34.1	4.2	2.00
Random, diverse-2	29.2	19.9	14.6	1.78
Shortest, diverse-2	30.0	21.5	14.2	1.78
Max symbolic, diverse-2	29.6	25.1	12.5	1.77
TEACHCE-AR	**35.8**	**41.1**	**3.8**	**1.75**

(b) Fresh ordering, $n = 120$

Order	Exact	Safe	Calls↓	Tok.↓
First CE / CEGIS	33.3	44.2	1.77	1.02k
Random	31.7	39.2	1.77	1.09k
Shortest	34.2	41.7	1.75	1.00k
Max symbolic	29.2	34.2	1.77	1.06k
Entropy gain	29.2	35.0	1.76	1.05k
Predict exact	37.5	45.8	1.69	0.99k
Predict progress	37.5	46.7	1.69	0.99k
TEACHCE-AR	**39.2**	**48.3**	**1.67**	**0.92k**

Table 3: **Sequential CEGIS comparison on fresh Datalog tasks.** All policies use the same verifier and state budget ($n = 256$). Outcomes are percentages. ↓ marks lower-is-better.

Policy	S@1	S@3	S@5	FinalErr↓	Calls↓
First-CE pair	52.3	73.0	77.3	22.4	2.31
Random pair	34.8	54.7	63.3	33.6	3.05
Complexity-aware pair	49.6	68.8	77.0	22.8	2.43
Two shortest	52.3	71.9	77.0	23.0	2.35
Max-information pair	39.1	63.7	75.4	23.2	2.72
Fixed TEACHCE + short	44.1	70.7	76.2	21.2	2.50
Sequential TEACHCE	**56.2**	**75.4**	**80.1**	**19.9**	**2.20**

These results answer RQ2 positively. Counterexample choice matters even when the available evidence, verifier, and model-call budget are held fixed. Predicting the learner's repair consequences provides a more useful ordering than ranking the evidence by formal informativeness alone, and adaptive reuse turns this ordering into better repair without requiring more computation.

5.3 RQ3: DOES LEARNER-SPECIFIC TEACHING HELP IN SEQUENTIAL CEGIS?

We next ask whether learner-specific counterexample selection remains useful when repairs are made over multiple CEGIS rounds. Unlike the one-step setting, each decision now changes the hypothesis and the verifier state seen by later rounds. Sequential TEACHCE therefore carries the accepted hypothesis and accumulated verifier evidence across calls, while its frozen router selects counterexample pairs from the observable disagreement pattern.

Table 3 compares six pair-selection strategies on 256 fresh Datalog tasks under the same state verifier and call budget. Sequential TEACHCE achieves the highest success rate at every evaluated horizon. Success@1 improves from 52.3% for the strongest competing selector to 56.2%, and Success@5 increases from 77.3% to 80.1%. It also leaves less residual semantic error and uses fewer calls on average. Every committed transition preserves the regression constraints.

These results answer RQ3 positively. Conditioning later choices on the evolving verifier state improves the repair trajectory as a whole, while the acceptance rule prevents unsuccessful revisions from overwriting previously verified behavior.

5.4 RQ4: DOES THE POLICY TRANSFER ACROSS MODELS AND REAL PROGRAMS?

We finally test whether the learner-specific policy transfers beyond the model and controlled tasks used for method development. We consider both a different LLM and executable student C programs.

Transfer across learner models. We first freeze the counterexample rankings and adaptive protocol, then evaluate them on a 240-state, 720-response GPT-5.6 Sol matrix. TEACHCE-AR reaches 38.3% exact recovery with 1.66 calls on average, improving exact recovery by 6.25 points over the corresponding one-call policy. Its accuracy is indistinguishable from repeated two-sample repair within uncertainty, while its average token use decreases from 1.30k to 1.08k. The broader static comparison in Table 4 shows the same pattern: across DeepSeek Flash and GPT-5.6 Sol, TEACHCE

Table 4: **Paired two-backbone comparison of 11 baselines and static TEACHCE.** ↓ marks lower-is-better. Each cell averages five repairs on 36 tasks. Choices are shared across backbones.

	DeepSeek Flash			GPT-5.6 Sol			
Teacher	Exact↑	Prog↑	Cat↓	Exact↑	Prog↑	Cat↓	Avg. rank↓
First CE (CEGIS) (Solar-Lezama et al., 2006)	20.0	−14.4	9.4	28.9	3.6	4.4	7.17
Random CE (Madaan et al., 2023)	16.1	−26.2	6.7	33.9	18.3	3.9	5.42
Shortest CE (Xia & Zhang, 2024)	18.9	8.1	8.3	28.3	10.3	3.9	6.17
Diverse CE Selection (Sener & Savarese, 2017)	16.7	−16.0	7.2	28.3	10.8	3.3	6.50
Learner-aware Elimination (Fan et al., 2018)	20.6	−76.1	10.6	33.9	−228.6	7.8	8.67
Max Symbolic Information (Liu et al., 2026)	20.6	−281.4	10.0	32.8	8.4	7.2	8.08
Entropy Gain (Gou et al., 2024)	21.1	−282.0	10.0	35.0	9.8	7.2	7.00
Oracle Elimination (Liu et al., 2026)	21.1	−346.2	11.1	28.3	4.9	6.1	9.00
Predicted Safe Uptake	17.8	−75.5	10.6	33.9	−214.5	5.6	8.25
Predicted Exact Recovery	17.2	−12.5	9.4	30.0	13.7	4.4	6.33
Predicted Progress	15.6	17.4	5.6	**36.1**	25.2	3.3	3.58
TEACHCE (Ours)	**22.8**	**25.4**	**5.0**	33.3	**30.4**	**2.2**	**1.83**

Table 5: **Blind repair of real C-Pack student programs.** All methods use DeepSeek Flash. Exact/M denotes exact repairs per million generated tokens.

Method	Exact ↑	Progress ↑	Calls ↓	Tok. ↓	Exact/M ↑
CEGIS	89.6	80.8	1.27	1.75k	517
Random	86.4	75.2	1.39	1.62k	532
Shortest	89.6	80.8	1.27	1.60k	561
Max-Information	89.6	80.8	1.27	1.60k	561
Repeated sampling	89.6	81.2	1.27	1.70k	526
Verifier reflection	84.8	66.0	1.27	1.99k	426
TEACHCE-AR	**90.4**	**84.0**	1.27	1.60k	561

attains the best average rank over the reported repair and safety metrics. More models are shown in Appendix D.3.

These results suggest that the useful structure captured by learner-specific selection is not tied to the capability level of the model used for development. A stronger learner changes the absolute probability of successful repair, but the benefit of reasoning about repair consequences remains.

Transfer to real program repair. We further evaluate blind repair on real C-Pack student programs. TEACHCE-AR achieves the highest exact-repair rate of 90.4% and the highest progress rate of 84.0%, while requiring only 1.27 calls and 1.60k tokens per task. Compared with standard CEGIS, TEACHCE-AR raises the exact-repair point estimate by 0.8 points and progress by 3.2 points at the same number of calls, while reducing token usage from 1.75k to 1.60k. Shortest and Max-Information use the same token budget as TEACHCE-AR, but reach 89.6% exact repair and 80.8% progress. Thus, combining learner-specific ranking with adaptive verifier-guided reuse improves repair quality without increasing the inference budget.

The broader real-program diagnostic in Figure 1 provides the stronger test of the underlying selection hypothesis. There, formal rank is only weakly related to safe repair on both C-Pack and IntroClass. In the larger 199-program replay (Table 17), learner-specific selection followed by a shortest fallback also improves full-suite repair over one-call maximum-information selection.

6 CONCLUSIONS

We show that formal informativeness and LLM teachability are distinct: a counterexample that strongly constrains the hypothesis space need not induce a better repair. TEACHCE addresses this gap by predicting learner-specific repair consequences, and TEACHCE-AR combines this ranking with verifier-guided adaptive reuse. Across controlled tasks, multiple LLMs, sequential CEGIS, and real program repair, the results support a broader principle: effective feedback selection should account not only for what the evidence reveals, but also for how the learner is likely to respond.

AI USE STATEMENT

In this work, generative AI tools were used to assist with language refinement, manuscript organization and presentation. AI tools were also used to improve the presentation of figures. All research hypotheses, methodological decisions, experimental implementations, executions, and scientific conclusions were independently evaluated and verified by the authors. No experimental results or empirical evidence were generated or fabricated by generative AI. All AI-assisted content was manually reviewed, and the authors take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide the complete algorithms, learner and selector configurations, hyperparameters, task construction and data-splitting procedures, statistical protocols, and representative repair prompts in the appendices. The controlled benchmarks are evaluated by exact execution over finite semantic universes, and all matched comparisons use identical learner, verifier, and compute settings unless stated otherwise.

REFERENCES

Alessandro Abate, Cristina David, Pascal Kesseli, Daniel Kroening, and Elizabeth Polgreen. Counterexample guided inductive synthesis modulo theories. In *Computer Aided Verification*, volume 10981 of *Lecture Notes in Computer Science*, pp. 270–288, 2018. doi: 10.1007/978-3-319-96145-3_15.

Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987. doi: 10.1016/0890-5401(87)90052-6.

Andrew Cropper and Sebastijan Dumančić. Inductive logic programming at 30: A new introduction. *Journal of Artificial Intelligence Research*, 74:765–850, 2022. doi: 10.1613/jair.1.13507.

Andrew Cropper and Rolf Morel. Learning programs by learning from failures. *Machine Learning*, 110(4):801–856, 2021.

Yang Fan, Fei Tian, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. Learning to teach. In *International Conference on Learning Representations*, 2018.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, et al. Critic: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pp. 57734–57811, 2024.

Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices*, 46(1):317–330, 2011.

Susmit Jha and Sanjit A Seshia. Are there good mistakes? a theoretical analysis of cegis. *arXiv preprint arXiv:1407.5397*, 2014.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Harshit Joshi, José Cambronero Sanchez, Sumit Gulwani, Vu Le, Gust Verbruggen, and Ivan Radiček. Repair is nearly generation: Multilingual program repair with llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 5131–5140, 2023.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024.

Jiaolong Kong, Xiaofei Xie, Mingfei Cheng, Shangqing Liu, Xiaoning Du, and Qi Guo. Contrastrepair: Enhancing conversation-based automated program repair via contrastive test case pairs. *ACM Transactions on Software Engineering and Methodology*, 34(8):1–31, 2025.

Louis Lalonde, Wassim Keddache, Thomas Perron Touchette, Leuson Da Silva, and Foutse Khomh. Revisiting feedback-driven llm code repair: A replication and exploratory java extension. *arXiv preprint arXiv:2609.00362*, 2026.

Claire Le Goues, Neal Holtschulte, Edward K Smith, Yuriy Brun, Premkumar Devanbu, Stephanie Forrest, and Westley Weimer. The manybugs and introclass benchmarks for automated repair of c programs. *IEEE Transactions on Software Engineering*, 41(12):1236–1256, 2015.

Hongyi Liu, Frederic Sala, Thomas Reps, and Adithya Murali. Counterexample guided learning in the large using reasoning agents. *arXiv preprint arXiv:2606.11521*, 2026.

David JC MacKay. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992.

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.

Farnam Mansouri, Yuxin Chen, Ara Vartanian, Jerry Zhu, and Adish Singla. Preference-based batch and sequential teaching: Towards a unified view of models. *Advances in neural information processing systems*, 32, 2019.

Pedro Orvalho, Mikoláš Janota, and Vasco Manquinho. C-pack of ipas: A c90 program benchmark of introductory programming assignments. In *Proceedings of the 5th ACM/IEEE International Workshop on Automated Program Repair*, pp. 14–21, 2024.

Pedro Orvalho, Mikoláš Janota, and Vasco Manquinho. Mentor: Fixing introductory programming assignments with formula-based fault localization and llm-driven program repair. *Journal of Systems and Software*, pp. 112690, 2025a.

Pedro Orvalho, Mikoláš Janota, and Vasco M Manquinho. Counterexample guided program repair using zero-shot learning and maxsat-based fault localization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 649–657, 2025b.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. Automatically correcting large language models: Surveying the landscape of diverse automated correction strategies. *Transactions of the Association for Computational Linguistics*, 12:484–506, 2024.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 60808–60838, 2025.

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.

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pp. 404–415, 2006.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *International conference on learning representations*, volume 2025, pp. 98190–98243, 2025.

Sebastian Tschiatschek, Ahana Ghosh, Luis Haug, Rati Devidze, and Adish Singla. Learner-aware teaching: Inverse reinforcement learning with preferences and constraints. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Chunqiu Steven Xia and Lingming Zhang. Automated program repair via conversation: Fixing 162 out of 337 bugs for $0.42 each using chatgpt. In *Proceedings of the 33rd ACM SIGSOFT international symposium on software testing and analysis*, pp. 819–831, 2024.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Yuze Zhao, Zhenya Huang, Yixiao Ma, Rui Li, Kai Zhang, Hao Jiang, Qi Liu, Linbo Zhu, and Yu Su. Repair: Automated program repair with process-based feedback. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 16415–16429, 2024.

Xiaojin Zhu. Machine teaching: An inverse problem to machine learning and an approach toward optimal education. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, pp. 4083–4087, 2015.

Xiaojin Zhu, Adish Singla, Sandra Zilles, and Anna N. Rafferty. An overview of machine teaching. *arXiv preprint arXiv:1801.05927*, 2018.

A Algorithms and Implementation Details

This appendix gives the full procedures for TeachCE, TeachCE-AR, and the sequential extension, followed by the implementation settings shared across the experiments.

A.1 Freeze-then-select

TeachCE learns a counterexample-selection policy from development tasks and then applies the resulting selector to unseen repair states. Cross-validation is grouped by task so that counterexamples from the same underlying repair problem do not appear in both the training and validation portions of a fold. Model selection is based on the downstream outcome of the counterexample chosen by each candidate policy rather than on predictor error alone.

Algorithm 1 Task-grouped fitting and frozen deployment of TeachCE.

Require: Development tasks $\mathcal{D}$ with tuples $(\phi_t(x), Z(x))$; candidate model families $\mathcal{M}$; acquisition weights $\mathcal{W}$
Ensure: Selector bundle B and one selected counterexample at deployment
Development phase
Partition $\mathcal{D}$ into task-grouped cross-validation folds
for all $(M, w) \in \mathcal{M} \times \mathcal{W}$ **do**
  for all cross-validation folds k **do**
    Fit every consequence head $\widehat{m}_j^{(-k)}$ outside fold k
    Score every validation candidate with $\widehat{A}_w^{(-k)}$ from Equation 7
    Select one candidate per validation task using a fixed tie-break
  end for
  Aggregate normalized progress and the ordered secondary metrics
end for
Choose $(M^\star, w^\star)$ by the pre-specified metric order
Refit all heads on $\mathcal{D}$ and construct $B = (\phi, M^\star, w^\star, \{\widehat{m}_j\}_j)$
Deployment phase
Require: Selector B, current hypothesis h_t, and candidate pool $\mathcal{C}_t$
for all $x \in \mathcal{C}_t$ **do**
  Compute $\phi_t(x)$ and consequence predictions $\{\widehat{m}_j(x)\}_j$
  Compute $\widehat{A}_{w^\star}(x)$
end for
$x_t \leftarrow \arg\max_{x \in \mathcal{C}_t} \widehat{A}_{w^\star}(x)$
Show $(x_t, y_t^\star)$ to the learner and return the revised hypothesis

Algorithm 2 Adaptive rollout reuse in TEACHCE-AR.

Require: Selector B, learner q_θ, current hypothesis h_t, labeled pool $\mathcal{C}_t$, regression suite R_t, and rollout cap K

```
Rank C_t by Equation 7 as x_(1), ..., x_(|C_t|)
R ← [ ]
for k = 1, ..., min(K, |C_t|) do
    Draw repair r_k ~ q_θ(· | h_t, x_(k))
    Execute r_k on x_(k), C_t, and R_t
    Append r_k and its verifier outcomes to R
    if r_k is valid, fixes x_(k), is pool-consistent, and regression-preserving then
        return r_k
    end if
end for
return the maximal repair in R under ≻_ver
```

The deployment features are computed before the learner is queried. They describe the current hypothesis, the candidate counterexample, its verifier label, symbolic scores, and candidate-pool statistics. Target-relative features are used only in the diagnostic ablations reported later; the prospective selection experiments use the observable feature contract.

A.2 ADAPTIVE ROLLOUT REUSE

A static selector determines which counterexample should be tried first, but the resulting LLM repair remains stochastic. TEACHCE-AR therefore executes each generated repair immediately and decides whether another counterexample is needed.

Repairs are compared by a fixed verifier preference $\succ_{\text{ver}}$. The ordering prioritizes syntactic validity, higher consistency with the labeled counterexample pool, lower regression rate, successful correction of the displayed counterexample, smaller absolute change in program size, and finally smaller resulting program size. This ordering is used only to choose a fallback when no rollout satisfies the early-stopping condition.

The primary TEACHCE-AR setting allows at most two distinct counterexamples. Importantly, every model call produces a candidate final repair: unsuccessful rollouts are retained rather than used only as probes. The same adaptive executor is used when comparing different counterexample orderings.

For a generated repair h', TEACHCE-AR evaluates three conditions. First, validity requires that the repaired program can be executed and passes the basic verifier checks. Second, pool consistency requires:

$$\forall (x, y) \in C_t, \quad h'(x) = y.$$

That is, the repair must satisfy all currently accumulated counterexample constraints. Third, regression preservation requires:

$$\forall z \in R_t, \quad h'(z) = h_t(z) = h^\star(z).$$

Therefore, an accepted repair must satisfy both accumulated feedback and previously correct behaviors.

A.3 SEQUENTIAL TEACHCE

For multi-round repair, the controller must also preserve information obtained in earlier rounds. We maintain the accepted hypothesis, accumulated labeled counterexamples, preservation examples, and recently rejected revisions.

The initial verifier pool is assigned one of three observable signatures: G if all disagreements require a $0 \to 1$ change, S if all require a $1 \to 0$ change, and Mix otherwise. The router uses a TEACHCE–SHORT pair for the two one-sided cases and a TWO-SHORT pair for mixed disagreement pools.

The sequential prompt extends the one-step repair prompt with the accumulated counterexamples, eight preservation examples, and up to three rejected revisions with verifier feedback. The target

Algorithm 3 Sequential TEACHCE with observable pair routing and accumulated verifier state.

Require: Selector B, router g, verifier V, accepted hypothesis h_1, regression suite R, and call budget T

$\mathcal{C}_1 \leftarrow V(h_1)$ ▷ labeled disagreements
$d \leftarrow \mathsf{G}$ if every $(x, y) \in \mathcal{C}_1$ has $(h_1(x), y) = (0, 1)$
    $d \leftarrow \mathsf{S}$ if every $(x, y) \in \mathcal{C}_1$ has $(h_1(x), y) = (1, 0)$; otherwise $d \leftarrow \mathsf{Mix}$
$a \leftarrow g(d)$, where $g(\mathsf{G}) = g(\mathsf{S}) =$ TEACHCE–SHORT and $g(\mathsf{Mix}) =$ TWO-SHORT
$\mathcal{M} \leftarrow \emptyset$ ▷ labeled counterexample memory
$\mathcal{F} \leftarrow [\,]$ ▷ rejected revisions
for $t = 1, \ldots, T$ **do**
    $\mathcal{C}_t \leftarrow \mathcal{C}_1$ if $t = 1$, otherwise $V(h_t)$
    Remove inputs already stored in $\mathcal{M}$
    if $a =$ TEACHCE–SHORT **then**
        $x_t \leftarrow \arg\max_{x \in \mathcal{C}_t} \widehat{A}(x)$ using B
        $z_t \leftarrow$ shortest opposite-label member, or shortest distinct member
    else
        $x_t \leftarrow$ shortest member of $\mathcal{C}_t$
        $z_t \leftarrow$ shortest opposite-label member, or shortest distinct member
    end if
    $P_t \leftarrow \{(x_t, y_t), (z_t, y'_t)\}$; omit z_t if unavailable
    $\mathcal{M} \leftarrow \mathcal{M} \cup P_t$
    Request r_t using h_t, $\mathcal{M}$, eight preservation examples from R, and the last three revisions in $\mathcal{F}$
    if r_t satisfies every example in $\mathcal{M}$ and preserves all of R **then**
        $h_{t+1} \leftarrow r_t$
    else
        Append r_t to $\mathcal{F}$
        $h_{t+1} \leftarrow h_t$
    end if
    if $V(h_{t+1})$ accepts **then**
        return h_{t+1}
    end if
end for
return the last accepted hypothesis h_{T+1}

program itself is not provided to the learner. A proposed revision is committed only when it satisfies all accumulated counterexamples and preserves the regression suite. The routing strategy in Algorithm 3 is a practical approximation of the ideal consequence-aware policy analyzed in Appendix B.5.

A.4 IMPLEMENTATION DETAILS

LLM backbones. DeepSeek V4 Flash is the primary learner for the controlled experiments and the public-program repair experiments. The paired transfer experiments additionally use GPT-5.6 Sol. To test whether the informativeness–teachability mismatch depends on a particular model or reasoning configuration, we further evaluate DeepSeek V4 Pro, Qwen3 235B Inst., and Gemini 3.6 Flash. The corresponding settings are non-thinking for DeepSeek V4 Flash and Qwen3, high reasoning for DeepSeek V4 Pro, minimal reasoning for Gemini 3.6 Flash, and low reasoning for GPT-5.6 Sol. All methods within the same comparison use the same learner, prompt, and decoding setting.

All LLM inference is performed through model-provider APIs. Controlled experiments use structured outputs containing a diagnosis, a revised executable rule, and a short change summary. Only the revised rule is used for evaluation. If the response cannot be parsed into the required format, one deterministic formatting retry is allowed.

Selector and hyperparameters. The consequence heads in the main observable TEACHCE selector are implemented with ExtraTrees models. Each repair consequence is predicted separately and

the predictions are combined by Equation 7. The default acquisition weights are fixed across all experiments. For clarity, the mapping between consequence heads and weights is:

Head	Z_{exact}	Z_{safe}	$\bar{G}$	Z_{reg}	Z_{cat}	D
w_j	1	0.25	0.5	-0.5	-2	-2

The symbolic informativeness bonus is controlled by $w_I = 0.1$ in all experiments unless otherwise specified. The 240-task observable matrix uses ten candidate counterexamples per repair state. The primary adaptive setting uses a rollout cap of $K = 2$ distinct counterexamples. Sequential experiments use a maximum of $T = 5$ learner calls, with eight preservation examples and at most three rejected revisions included in the repair context.

The initial controlled uptake audit uses three independent repairs per counterexample. Main paired comparisons use five repairs per selected counterexample, while the high-replication reliability analysis uses 20 repairs per counterexample. The deployed consequence predictor contains six heads: exact recovery, safe uptake, normalized semantic progress, regression risk, catastrophic failure risk, and damage score.

Baselines. For static counterexample selection, we compare against the first counterexample used by standard CEGIS, random selection, the shortest counterexample, maximum symbolic information, entropy gain, oracle-conditioned elimination, and pool diversity. We also include learner-side selectors that predict only safe uptake, exact recovery, or semantic progress, together with learner-aware elimination. These baselines separate the benefit of joint consequence prediction from that of any single predicted outcome.

For adaptive repair, we compare TEACHCE-AR with repeated sampling, verifier reflection, and strategies that spend the same call budget on two distinct counterexamples. When isolating the effect of ordering, random, shortest, maximum-information, entropy-gain, and learner-specific rankings all use the same adaptive executor. The sequential comparison includes first-pair, random-pair, complexity-aware, two-shortest, maximum-information, and fixed TEACHCE-plus-shortest policies.

Executable evaluation. All controlled hypotheses are evaluated by exact execution rather than by model-based judging. Datalog is evaluated over all 512 Boolean worlds. Regex is evaluated over all 5,461 strings of length at most six in the fixed alphabet, and the list DSL is evaluated over its complete bounded input universe. Invalid completed outputs count as failed repairs.

The real-program experiments use C-Pack and IntroClass. The complete current C90 program is provided to the learner, and returned programs are compiled and evaluated using the benchmark-native execution semantics and test suites. All methods within a comparison use the same verifier and model-call budget.

B THEORETICAL ANALYSIS

We provide the formal results underlying the decision view in Section 4. The analysis has three purposes. First, it shows that symbolic informativeness can be arbitrarily misaligned with the repair induced by a learner, even under a simple monotone update model. Second, it characterizes the ideal learner-specific teacher and the effect of prediction error on the resulting decision. Finally, it extends the same view to cost-aware and finite-horizon repair.

Throughout this section, the selection-time candidate set $\mathcal{C}_t$ is finite and nonempty, all outcome variables are integrable, the utility weights are fixed, and the normalized symbolic score $\widetilde{E}_t(x)$ is measurable with respect to the information $\mathcal{F}_t$ available before the learner call. For each outcome $j \in \mathcal{J}$, let

$$m_j(x) = \mathbb{E}[Z_j(x) \mid \mathcal{F}_t].$$

We define the corresponding population acquisition function as

$$A^\star_w(x) = \sum_{j \in \mathcal{J}} w_j m_j(x) + w_I \widetilde{E}_t(x). \tag{10}$$

B.1 Failure of Symbolic Informativeness

The central distinction in this paper is that a counterexample can strongly constrain the hypothesis space while still being difficult for a particular learner to use. The following construction shows that this mismatch can be arbitrarily large even for a simple learner whose probability of successful uptake decreases monotonically with repair difficulty.

Proposition 3 (Arbitrarily poor selection under a monotone logistic updater)**.** *For every $\epsilon \in (0, 1)$, there exists a two-counterexample teaching problem in which the unique symbolic-score maximizer attains less than an ϵ fraction of the best expected one-step progress. This holds for a two-state learner that returns either h_t or $h^\star$, with uptake probability*

$$q_\beta(x) = \sigma(-\beta c(x)),$$

for a difficulty $c(x) \in [0, 1]$.

Proof. Fix $\epsilon \in (0, 1)$. Let the evaluation universe be $\mathcal{X} = \{x_1, x_2\}$, with target and current predictions

$$h^\star = (0, 0), \qquad h_t = (1, 1)$$

on (x_1, x_2). Hence both inputs are valid counterexamples.

Let the current version space contain five hypotheses with prediction vectors

$$(0, 0),\ (0, 1),\ (1, 1),\ (1, 1),\ (1, 1),$$

where hypotheses with the same prediction vector are distinguished elsewhere in their domain. Under Equation 1,

$$E_t(x_1) = 2, \qquad E_t(x_2) = 1,$$

so x_1 is the unique symbolic-score maximizer.

Assign difficulties

$$c(x_1) = 1, \qquad c(x_2) = 0.$$

After observing either candidate, the learner returns $h^\star$ with probability $q_\beta(x) = \sigma(-\beta c(x))$ and otherwise leaves the current hypothesis unchanged. Successful uptake therefore removes both semantic errors, whereas failed uptake removes none, giving

$$\mathbb{E}[G_t(x)] = 2q_\beta(x).$$

Thus

$$\frac{\mathbb{E}[G_t(x_1)]}{\mathbb{E}[G_t(x_2)]} = \frac{\sigma(-\beta)}{\sigma(0)} = \frac{2}{1 + e^\beta}.$$

Choosing

$$\beta > \log(2/\epsilon - 1)$$

makes this ratio smaller than ϵ, although x_1 remains the unique symbolic maximizer.

Hence symbolic informativeness and expected repair quality can be arbitrarily misaligned even when the learner has only two possible next states and follows the same monotone logistic update rule for every candidate. □

This result does not rely on regressions, candidate-specific update rules, or a large hypothesis space. The gap arises simply because hypothesis-space elimination and learner uptake measure different quantities.

B.2 Bayes-Optimal Learner-Specific Teaching

We next characterize the ideal one-step teacher when the conditional repair consequences are known. Recall that $L_w(x)$ denotes the utility associated with presenting counterexample x.

Theorem 4 (Bayes-optimal one-step teacher)**.** *Any maximizer of $A^\star_w$ maximizes*

$$\mathbb{E}[L_w(x) \mid \mathcal{F}_t]$$

over $x \in \mathcal{C}_t$. Consequently, the plug-in rule in Equation 7 is Bayes optimal when $\widehat{m}_j(x) = m_j(x)$ for every outcome head and candidate.

Proof. Since $\widetilde{E}_t(x)$ is $\mathcal{F}_t$-measurable, linearity of conditional expectation gives

$$\begin{aligned}\mathbb{E}[L_w(x) \mid \mathcal{F}_t] &= \sum_{j \in \mathcal{J}} w_j \mathbb{E}[Z_j(x) \mid \mathcal{F}_t] + w_I \widetilde{E}_t(x) \\ &= \sum_{j \in \mathcal{J}} w_j m_j(x) + w_I \widetilde{E}_t(x) \\ &= A^\star_w(x).\end{aligned}$$

Therefore any maximizer of $A^\star_w$ maximizes conditional expected utility. If $\widehat{m}_j(x) = m_j(x)$ for all j and x, then $\widehat{A}_w(x) = A^\star_w(x)$ for every candidate, so the plug-in rule selects the same optimum. □

The formulation also contains a simple learner-aware variant of symbolic elimination as a special case.

Corollary 5 (Uptake-weighted symbolic elimination)**.** *Let B_x denote successful uptake of candidate x, with conditional probability*

$$U(x) = \Pr(B_x \mid \mathcal{F}_t).$$

Under the binary progress model

$$G_t(x) = E_t(x)\mathbf{1}[B_x],$$

the expected progress is

$$\mathbb{E}[G_t(x) \mid \mathcal{F}_t] = U(x)E_t(x).$$

If presenting x also incurs a nonnegative regression cost C_x with conditional mean $R(x)$, then a teacher maximizing

$$\mathbb{E}[G_t(x) - \lambda C_x \mid \mathcal{F}_t]$$

selects

$$\arg\max_{x \in \mathcal{C}_t} \{U(x)E_t(x) - \lambda R(x)\}.$$

Proof. Because $E_t(x)$ is known at selection time,

$$\mathbb{E}[G_t(x) \mid \mathcal{F}_t] = E_t(x)\mathbb{E}[\mathbf{1}[B_x] \mid \mathcal{F}_t] = E_t(x)U(x).$$

Linearity then gives

$$\mathbb{E}[G_t(x) - \lambda C_x \mid \mathcal{F}_t] = U(x)E_t(x) - \lambda R(x),$$

and maximizing over the finite candidate set proves the result. □

This special case makes explicit why symbolic information alone is insufficient: its value is modulated by the probability that the current learner can successfully act on it.

B.3 Plug-in Regret Under Prediction Error

In practice, TeachCE does not know the conditional means $m_j(x)$ and instead uses the learned estimates $\widehat{m}_j(x)$. The next result relates the quality of these predictions directly to the quality of the selected counterexample.

Theorem 6 (Plug-in acquisition regret)**.** *Assume that, for every $x \in \mathcal{C}_t$ and $j \in \mathcal{J}$,*

$$|\widehat{m}_j(x) - m_j(x)| \le \eta_j.$$

Let

$$x^\star \in \arg\max_x A^\star_w(x), \qquad \widehat{x} \in \arg\max_x \widehat{A}_w(x).$$

Then

$$A^\star_w(x^\star) - A^\star_w(\widehat{x}) \le 2 \sum_{j \in \mathcal{J}} |w_j| \eta_j. \tag{11}$$

Proof. Let

$$\Delta = \sum_{j \in \mathcal{J}} |w_j| \eta_j.$$

The symbolic term is identical in $A^\star_w$ and $\widehat{A}_w$, so

$$|\widehat{A}_w(x) - A^\star_w(x)| \le \Delta$$

for every candidate. Therefore

$$\begin{aligned} A^\star_w(x^\star) - A^\star_w(\widehat{x}) = &\left[A^\star_w(x^\star) - \widehat{A}_w(x^\star)\right] \\ &+ \left[\widehat{A}_w(x^\star) - \widehat{A}_w(\widehat{x})\right] \\ &+ \left[\widehat{A}_w(\widehat{x}) - A^\star_w(\widehat{x})\right]. \end{aligned}$$

The first and third terms are each at most Δ, while the middle term is nonpositive because $\widehat{x}$ maximizes $\widehat{A}_w$. Hence

$$A^\star_w(x^\star) - A^\star_w(\widehat{x}) \le 2\Delta,$$

which proves Equation 11. □

The bound is deliberately agnostic to the choice of consequence predictor. It only states that improvements in consequence estimation translate directly into a tighter decision-quality guarantee.

B.4 Cost-Aware Repair Planning

The same decision view can include the cost of obtaining a repair. This is useful for adaptive policies such as TeachCE-AR, where an action may include both a counterexample choice and a rollout allocation.

Corollary 7 (Cost-aware repair-plan selection)**.** *Let $\mathcal{A}_t$ be a finite set of repair plans. A plan may specify a counterexample, a rollout allocation, and a stopping rule. Let $c(a)$ denote its selection-time measurable cost and define*

$$A^\star_{w,\kappa}(a) = \sum_{j \in \mathcal{J}} w_j m_j(a) - \kappa c(a).$$

Any maximizer of $A^\star_{w,\kappa}$ maximizes conditional expected utility net of cost.

If the consequence means satisfy the error bounds of Theorem 6, then the corresponding plug-in plan has regret at most

$$2 \sum_{j \in \mathcal{J}} |w_j| \eta_j.$$

If cost is also estimated with uniform error

$$|\widehat{c}(a) - c(a)| \le \xi,$$

the bound becomes

$$2 \left(\sum_{j \in \mathcal{J}} |w_j| \eta_j + |\kappa| \xi \right).$$

Proof. Because $c(a)$ is measurable at selection time, $A^\star_{w,\kappa}(a)$ is exactly the conditional expected utility of plan a minus its weighted cost. With exact cost, the cost term is shared by the population and plug-in objectives, so the proof of Theorem 6 applies directly over $\mathcal{A}_t$.

If cost is estimated, then for every a,

$$|\widehat{A}_{w,\kappa}(a) - A^\star_{w,\kappa}(a)| \le \sum_{j \in \mathcal{J}} |w_j| \eta_j + |\kappa| \xi.$$

Applying the same two-sided comparison as in Theorem 6 gives twice this quantity. □

Thus adaptive repair can be viewed as selecting among repair plans rather than individual counterexamples, with model calls or token usage entering the objective as an explicit cost.

B.5 FINITE-HORIZON EXTENSION

The one-step analysis extends to sequential repair when counterexample choices change the hypothesis and verifier state seen in later rounds. We state the result for a generic finite-horizon verifier process; the state may include the accepted hypothesis, accumulated evidence, and any other observable history used by the controller.

Theorem 8 (Finite-horizon value-error regret)**.** *Consider a finite-horizon verifier process with states s, finite action sets, and rewards that may include model-call or token costs. Let $Q_t^\star(s,a)$ denote the optimal action value at round t. Suppose*

$$|\widehat{Q}_t(s,a) - Q_t^\star(s,a)| \le \eta_t$$

uniformly over states and actions, and let

$$\widehat{\pi}_t(s) \in \arg\max_a \widehat{Q}_t(s,a).$$

Then, for every initial state s_1,

$$V_1^\star(s_1) - V_1^{\widehat{\pi}}(s_1) \le 2\sum_{t=1}^{T} \eta_t. \tag{12}$$

Proof. Fix a state s at round t. Let

$$a^\star \in \arg\max_a Q_t^\star(s,a), \qquad \widehat{a} = \widehat{\pi}_t(s).$$

By greediness of $\widehat{a}$ and the uniform approximation bound,

$$\begin{aligned} Q_t^\star(s,a^\star) - Q_t^\star(s,\widehat{a}) &\le \left[Q_t^\star(s,a^\star) - \widehat{Q}_t(s,a^\star)\right] \\ &\quad + \left[\widehat{Q}_t(s,\widehat{a}) - Q_t^\star(s,\widehat{a})\right] \\ &\le 2\eta_t. \end{aligned}$$

Let

$$\delta_t = \sup_s \left[V_t^\star(s) - V_t^{\widehat{\pi}}(s)\right], \qquad \delta_{T+1} = 0.$$

For the action $\widehat{a}$, the immediate loss relative to the optimal action is at most $2\eta_t$. Following the transition induced by $\widehat{a}$, the additional loss from using $\widehat{\pi}$ instead of the optimal continuation is at most δ_{t+1}. Hence

$$\delta_t \le 2\eta_t + \delta_{t+1}.$$

Backward induction gives

$$\delta_1 \le 2\sum_{t=1}^{T} \eta_t,$$

which proves Equation 12. □

The theorem gives the sequential analogue of Theorem 6: approximation error in the learner-specific action values accumulates at most additively across the repair horizon. It does not require the sequential controller to use a particular state representation or routing rule.

B.6 RELATIONSHIP BETWEEN THE THEORETICAL POLICY AND SEQUENTIAL TEACHCE

The finite-horizon analysis studies an ideal learner-specific teaching policy that selects actions according to their estimated downstream utility. Specifically, the theoretical policy assumes access to an action-value function $Q_t(s,a)$ and selects

$$a_t = \arg\max_a \hat{Q}_t(s,a).$$

In practice, estimating the full multi-step value function is expensive. Therefore, Sequential TEACHCE adopts a tractable approximation. It uses the one-step consequence acquisition function

$\hat{A}_w(x)$ when the disagreement structure is suitable for consequence prediction, and switches to heuristic counterexample ordering in mixed disagreement regimes.

Therefore, Theorem 8 should be interpreted as a characterization of the ideal learner-specific sequential teaching problem. It motivates why accurate consequence estimation can improve sequential decision quality.

C Experimental Protocol and Statistical Analysis

This section summarizes the experimental design shared by the analyses that follow. Model configurations and the implementation of TeachCE are given in Appendix A; here we focus on data separation, repeated sampling, and statistical inference.

C.1 Controlled task splits

The large observable-only study contains 240 previously unseen repair states, with 80 tasks from each of Datalog, Regex, and the list DSL. Each state contains ten candidate counterexamples. Datalog and list-DSL tasks exclude semantic target–current pairs used during development. Regex additionally deduplicates target and current behavior over its complete finite evaluation universe.

The independent Regex confirmation contains 120 tasks balanced across six repair classes and 2,400 executable counterexamples. Its target–current pairs are disjoint from the 96 Regex development tasks. The controlled evaluation sets are fixed before model calls: Datalog is evaluated over all 512 Boolean worlds, Regex over all 5,461 strings of length at most six in the fixed alphabet, and the list DSL over its complete bounded universe.

For the high-replication analysis, we select 12 tasks per controlled family and ten counterexamples per task, giving 360 counterexamples. Selection is stratified by repair class, symbolic rank, and input complexity without using repair outcomes. Each counterexample receives 20 independent repairs. This subset is used to distinguish instability caused by finite sampling from a genuine mismatch between symbolic score and learner response.

C.2 Repeated repairs and statistical inference

The initial uptake audit uses three independent repairs per counterexample. Fresh selection comparisons use five repairs for each selected counterexample unless otherwise stated. Sequential experiments evaluate one trajectory per task because later verifier states depend on earlier model outputs.

Confidence intervals are obtained by resampling tasks rather than individual LLM calls. We use 10,000 task-level bootstrap resamples for the reported intervals and paired sign-flip tests for paired policy comparisons. For the high-replication analysis, we additionally use repeated subsampling at 3, 5, and 10 repairs and an empirical-Bayes random-effects model to estimate counterexample-level response variation.

Invalid completed outputs count as learner failures. A formatting error is given one deterministic retry that requests only the required response schema. Transport failures are excluded from learner-outcome statistics.

C.3 Matched-compute evaluation

Comparisons intended to isolate counterexample selection use the same learner, repair prompt, verifier, and number of scored repairs. Adaptive policies additionally share the same rollout cap and acceptance rule. This prevents a selection policy from gaining an advantage simply by receiving more repair attempts.

Table 6 reports the deployment cost of the primary matched-compute comparisons, including formatting retries, token usage, and recorded API cost. Calls include formatting retries and token counts include both the original and retry attempts.

Table 6: **Matched deployment compute for the primary comparisons.** Calls include deterministic format retries; token totals include both the initial and retry attempts. Cost uses the recorded DeepSeek prices.

Round	Policy	Calls	Tokens (k)	Cost ($)	Exact/Mtok
Datalog 1	Primary comparator	829	355.1	0.054	566
	TEACHCE	829	354.7	0.054	581
Datalog 2	Primary comparator	820	337.8	0.050	832
	TEACHCE	808	317.6	0.045	838
Regex	Primary comparator	721	520.1	0.100	154
	TEACHCE	776	645.8	0.128	135

Every row contains the same number of scored repairs: 800 in each Datalog round and 600 in Regex. Median end-to-end TEACHCE selection overhead is 3.4, 5.4, and 3.9 ms per task, respectively, on the audit machine.

D EXTENDED ANALYSIS OF THE INFORMATIVENESS–TEACHABILITY MISMATCH

The main paper shows that symbolic informativeness is only weakly related to the repair induced by an LLM. This section tests alternative explanations for that result: prompt choice, limited replication, learner capability, the particular symbolic score, and variation in repair structure.

D.1 ROBUSTNESS TO REPAIR PROMPTING

One possibility is that symbolic rank appears uninformative only because the default prompt encourages unconstrained revisions. We therefore repeat the Datalog audit with three repair instructions: the standard prompt, a minimal-edit instruction, and a diagnose–revise–verify instruction. Table 7 reports the resulting symbolic–progress association, teaching-rank inversions, and repair outcomes under the three prompting variants.

Table 7: **Prompt perturbation on the frozen Datalog subset.** Only the repair instruction changes; tasks, candidate choices, learner, decoding, and three samples per CE are fixed. ρ is the within-task rank correlation between symbolic elimination and realized progress. Policy entries are exact recovery / raw normalized progress / safe uptake (%).

Instruction	ρ	Inversion	Max symbolic	TEACHCE
Standard	-0.11	56.2	19.4/24.3/27.8	27.8/40.3/52.8
Minimal edit	-0.09	56.9	16.7/14.6/25.0	16.7/26.4/41.7
Verify then edit	-0.12	56.8	5.6/4.9/13.9	16.7/18.1/41.7

The symbolic–progress association remains close to zero under all three prompts, and teaching-rank inversions remain frequent. Prompting changes the absolute probability of successful repair, but does not restore the ordering predicted by symbolic informativeness. The mismatch is therefore not an artifact of one repair instruction.

D.2 RELIABILITY UNDER INCREASED REPLICATION

The weak symbolic relationship could also arise if three repairs per counterexample are insufficient to estimate learner response. The high-replication subset addresses this directly by collecting 20 repairs for each of 360 counterexamples.

As reported in the main paper, counterexample-level outcome estimates become substantially more stable as replication increases, while the association between symbolic rank and latent repair progress remains near zero. Thus, additional sampling reduces measurement noise without recovering the ordering predicted by symbolic information.

Table 9: **Stronger symbolic scores do not recover learner-specific teaching rank.** Within-task Spearman correlation between each exact score and mean realized progress; brackets are 95% task-bootstrap intervals.

Score	Datalog	Regex	List DSL
Maximin split	+0.02 [-0.04,+0.08]	-0.08 [-0.13,-0.02]	+0.07 [+0.00,+0.14]
Entropy gain	+0.02 [-0.04,+0.08]	-0.08 [-0.13,-0.02]	+0.07 [+0.00,+0.14]
Oracle-conditioned	+0.27 [+0.18,+0.35]	-0.35 [-0.43,-0.27]	+0.06 [-0.01,+0.13]

D.3 Generalization across learner models

We next ask whether the mismatch is specific to a particular model family or reasoning configuration. The same counterexamples are evaluated with DeepSeek V4 Flash, DeepSeek V4 Pro, Qwen3 235B Inst., Gemini 3.6 Flash, and GPT-5.6 Sol. Table 8 compares repair outcomes and the symbolic–progress association across the five learner settings.

Table 8: **The evidence–learner mismatch generalizes across models and task families.** Safe uptake / exact recovery (%) use the same 120 counterexamples and five repairs per family. Valid is final parse-valid output (%). ρ is the pooled noise-corrected within-task association between symbolic elimination and latent progress; C-TRIR is the confident teaching-rank inversion rate (%).

Model	Setting	Valid	Datalog	Regex	List DSL	ρ [95% CI]	C-TRIR
DeepSeek V4 Flash	non-thinking	94.9	27.2 / 16.5	37.0 / 14.3	26.8 / 21.5	-0.01 [-0.13,+0.12]	58.2
DeepSeek V4 Pro	high reasoning	93.9	22.2 / 17.5	31.3 / 14.2	51.8 / 48.7	-0.00 [-0.13,+0.13]	47.1
Qwen3 235B Inst.	non-thinking	96.0	9.5 / 9.2	23.5 / 6.2	38.2 / 33.3	+0.02 [-0.13,+0.16]	48.9
Gemini 3.6 Flash	minimal reasoning	99.6	9.3 / 7.0	34.3 / 18.3	47.5 / 41.8	+0.07 [-0.06,+0.20]	49.2
GPT-5.6 Sol	low reasoning	98.8	22.0 / 17.7	38.0 / 19.0	61.3 / 53.8	-0.00 [-0.14,+0.13]	46.0

The subset and analysis were frozen before external-model calls. Invalid completed outputs remain learner failures; transport failures are not scored. This table tests mechanism and task-family generalization, not cross-model transfer of the TeachCE selector.

Absolute repair success varies substantially across models and task families, as expected. In contrast, the within-task association between symbolic elimination and realized progress remains close to zero across all five model settings. Increasing learner capability can therefore improve the chance of successful repair without making symbolic informativeness a reliable proxy for which counterexample that learner will use best.

D.4 Alternative symbolic scores

The main analysis uses a maximin version-space split. To test whether the result depends on that particular choice, we also evaluate entropy gain and oracle-conditioned elimination. Table 9 compares the within-task association between each symbolic criterion and realized progress across Datalog, Regex, and List DSL.

Maximin and entropy produce essentially the same ordering in the binary candidate pools. Oracle-conditioned elimination changes the ranking more substantially, but its relation to realized progress is positive in Datalog, negative in Regex, and close to zero in the list DSL. No tested symbolic criterion therefore provides a domain-stable approximation to learner-specific teaching value.

D.5 Repair structure explains more than symbolic rank

The failure of symbolic ranking is not random. Repair success varies strongly with the semantic transition required from the learner. In particular, specializing and generalizing repairs can have markedly different uptake rates even when their symbolic scores are similar.

Figure 5 summarizes both the repair-by-complexity uptake patterns and the corresponding mixed-effect estimates. The mixed-effect analysis gives the same qualitative picture as the raw repair-class statistics. Required repair direction and input complexity explain substantial variation in safe uptake,

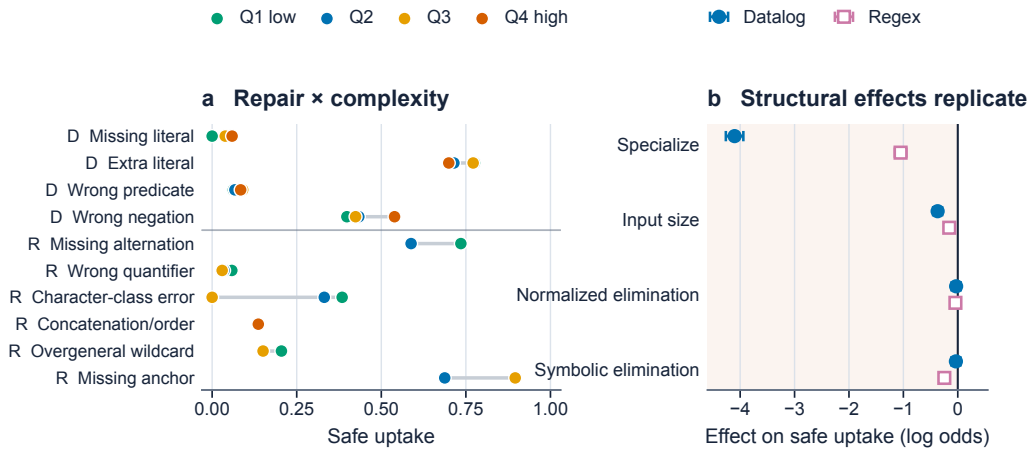


Figure 5: **Uptake depends on repair structure.** (a) Safe uptake across repair classes and input-complexity quartiles. (b) Fixed effects from family-specific mixed logistic models; whiskers denote 95% variational posterior intervals.

whereas symbolic elimination contributes little additional explanatory power once these structural factors are accounted for. This supports the interpretation that teachability is a property of the interaction between evidence and learner, rather than a noisy version of symbolic informativeness.

D.6 Representative teaching-rank inversion

A concrete example helps illustrate what the aggregate statistics measure. Table 10 compares two valid counterexamples from the same Datalog repair state and illustrates a representative teaching-rank inversion. The counterexample favored by symbolic elimination induces a repair with substantial regression, whereas the lower-ranked counterexample reliably recovers the intended rule.

Table 10: **A teaching-rank inversion on a fresh Datalog task.**

	Max elimination	TeachCE
Current rule	`target(X) :- p2(X), p5(X), not p7(X), p8(X).`	
Target rule	`target(X) :- p2(X), not p7(X), p8(X).`	
Counterexample	`{p0, p1, p2, p8}`	`{p2, p8}`
Oracle label	true	true
Maximin split score	213	189
Median-outcome repair	`target(X) :- p2(X), not p5(X), p8(X).`	`target(X) :- p2(X), not p7(X), p8(X).`
Displayed case fixed	✓	✓
Regression-free	×	✓
Regressions	16	0
Normalized progress	-100.0%	+100.0%

Across five repairs, safe uptake is 0/5 versus 5/5 and mean normalized progress differs by 180.0 points.

The example shows how a teaching-rank inversion can arise even when both counterexamples are valid and both displayed cases are locally corrected.

E Extended Results for Learner-Specific Selection

The previous section establishes that counterexample quality depends on the learner response. We now ask whether that dependence is predictable enough to improve counterexample choice. Table 11

provides the full fresh observable-only comparison, including pooled outcomes and domain-wise exact recovery.

E.1 OBSERVABLE CONSEQUENCE PREDICTION

We first evaluate learner-response prediction on 240 fresh repair states from the three controlled task families. Each state contains ten candidate counterexamples, and all policies are evaluated on held-out repairs.

Table 11: **Fresh 240-task observable-only confirmation.** All policies are frozen before five held-out repairs per selected CE. Exact and bounded progress (BProg) are higher better; catastrophic regression (Cat) is lower better. The final three columns decompose exact recovery by domain.

	Pooled (240 tasks)			Exact recovery		
Teacher	Exact↑	BProg↑	Cat↓	Datalog	Regex	List DSL
First CE / CEGIS	20.1	11.2	11.4	22.5	5.8	32.0
Random CE	20.9	14.9	11.8	21.7	5.0	36.0
Shortest CE	21.4	11.0	12.1	26.0	6.2	32.0
Max symbolic information	21.0	14.4	11.2	23.8	4.8	34.5
Entropy gain	21.0	14.3	11.2	23.8	4.8	34.5
Oracle-conditioned elimination	22.4	7.3	18.2	25.0	5.5	36.8
Predicted exact recovery	25.3	17.9	11.1	<u>30.8</u>	<u>6.0</u>	39.2
Predicted bounded progress	25.2	<u>27.2</u>	5.7	29.3	**7.0**	39.5
TEACHCE (no probe)	24.6	25.9	<u>5.0</u>	<u>30.8</u>	4.8	38.2
Probe pool consistency	26.4	17.2	14.7	28.7	4.8	45.8
Probe-predicted exact	<u>29.1</u>	24.5	8.6	29.5	5.0	**52.8**
TEACHCE-Probe (ours)	**29.5**	**31.0**	**4.8**	**31.0**	5.3	<u>52.2</u>

Percentage points. Bold and underline mark the best and second-best values, with ties preserved. The primary paired difference in exact recovery is $+4.92$ points over TEACHCE without a probe (95% CI $[2.08, 7.83]$); the compute-matched difference over probe pool consistency is $+3.08$ points ($[0.67, 5.50]$).

Predicting repair consequences improves selection over purely symbolic criteria. Predictors of exact recovery or bounded progress already provide useful signal, while combining multiple consequences gives a better balance between recovery and catastrophic failure. The higher-compute probe variants in the same table are included only as a diagnostic of available predictive headroom.

E.2 ADAPTIVE REUSE UNDER A MATCHED CALL BUDGET

We next isolate the effect of counterexample ordering when every policy uses the same adaptive executor. Candidates are tried under a cap of two distinct counterexamples, and the verifier and stopping rule are identical across policies. Table 12 reports the full outcome breakdown for this matched adaptive comparison, including bounded progress, regression, and catastrophic regression.

Table 12: **Full fresh ordering outcomes.** All policies use the same adaptive cap of two and the same observable acceptance and final-selection rule. Main-table efficiency columns are supplemented here by bounded progress (BProg), regression (Reg.), and catastrophic regression (Cat.). All outcome columns are percentages.

Order	Exact	BProg	Safe	Reg.	Cat.	Calls
First CE / CEGIS	33.3	30.7	44.2	43.3	11.7	1.77
Random	31.7	27.8	39.2	52.5	14.2	1.77
Shortest	34.2	31.7	41.7	45.0	10.8	1.75
Max symbolic information	29.2	24.9	34.2	45.8	13.3	1.77
Entropy gain	29.2	25.3	35.0	45.8	13.3	1.76
Predicted exact recovery	37.5	36.4	45.8	42.5	12.5	1.69
Predicted bounded progress	37.5	33.1	46.7	45.0	12.5	1.69
TEACHCE-AR	39.2	32.2	48.3	41.7	14.2	1.67

The learner-specific ordering attains the highest exact recovery and safe uptake while requiring fewer calls on average. Because the executor is fixed, these differences cannot be attributed to a more permissive verifier or to additional repair attempts. Predicting exact recovery or progress alone captures part of the gain, but remains below the joint consequence-based ordering.

E.3 WHICH FEATURES CARRY THE SELECTION SIGNAL?

The strongest controlled gains occur when features expose information about the semantic repair required by the candidate. We therefore separate this diagnostic setting from fully observable deployment. Table 13 isolates the role of feature access on fresh Datalog tasks by comparing target-relative, reduced, and observable-only feature regimes.

Table 13: **The fresh Datalog gain is not explained by distractor count alone.** All selectors are fit on the same 100 development tasks and evaluated on the same 160 fresh tasks with five repairs per choice. ΔProg is relative to max symbolic information; brackets are 95% paired task-bootstrap intervals. Values are percentages.

Feature regime	Exact	Progress	Safe	ΔProg
Full target-relative	28.1	34.8	48.4	+8.15 [+3.72,+12.77]
Full w/o distractor count	27.9	34.4	47.8	+7.76 [+3.29,+12.33]
Distractor count only	27.1	30.7	41.1	+4.06 [-1.25,+9.52]
Observable only	26.8	31.5	45.8	+4.86 [+0.81,+9.08]
Max symbolic information	22.0	26.6	34.8	–

Removing individual target-relative features leaves most of the controlled gain intact, while a distractor-count-only selector explains substantially less. Observable features retain some predictive signal, but the gap between the two regimes shows that inferring the required semantic transition from selection-time information remains difficult.

F SEQUENTIAL CEGIS

The one-step experiments evaluate a single teaching decision. We next test whether learner-specific feedback remains useful when each accepted repair changes the hypothesis and verifier state seen in later rounds.

F.1 FROM MYOPIC SELECTION TO OBSERVABLE ROUTING

Early development experiments showed that simply carrying counterexample history forward was insufficient: avoiding repeated inputs did not reliably improve longer-horizon recovery. We therefore augmented the controller with accumulated labeled counterexamples, preservation constraints, and

Table 14: **Fresh multi-step validation with observable routing.** Success is exact recovery within the call budget; FinalErr is remaining semantic error divided by initial error, Reg. is the rate of state transitions that increase semantic error, and Calls is the mean number of model calls before success or budget exhaustion. The router reads only the initial verifier-pool disagreement directions. 256 held-out Datalog tasks, one trajectory per policy. All columns except Calls are percentages.

Policy	S@1	S@3	S@5	FinalErr	Reg.	Calls
Two-shortest + state verifier	52.3	71.9	77.0	23.0	0.0	2.35
Sequential TEACHCE	56.2	75.4	80.1	19.9	0.0	2.20

rejected revisions, and commit a revision only when it remains consistent with the accumulated verifier state.

A second development stage showed that a fixed learner-specific order was not uniformly preferable across verifier states. This motivated the observable router used in Algorithm 3: one-sided disagreement pools use a TEACHCE–SHORT pair, while mixed pools use two short counterexamples with opposing labels.

These development experiments were used only to choose the state representation and router. We therefore omit their intermediate performance tables and report the final policy on a new task set below.

F.2 FRESH MULTI-STEP VALIDATION

The final router is evaluated on 256 new Datalog tasks with the same accumulated verifier state and a five-call budget for every policy. Table 14 compares Sequential TEACHCE with the controller-matched two-shortest policy across recovery horizons, residual error, regression, and model calls.

Sequential TEACHCE improves success at every reported horizon. Exact recovery increases from 52.3% to 56.2% after one call and from 77.0% to 80.1% within five calls relative to the controller-matched two-shortest policy. It also leaves less residual semantic error and uses fewer learner calls on average.

The zero regression rate in both policies reflects the commit rule: revisions that violate accumulated evidence or previously correct behavior are rejected and do not replace the accepted hypothesis. The sequential gain therefore comes from choosing more useful repair actions within the same verifier-controlled trajectory.

G TRANSFER ACROSS MODELS AND REAL PROGRAMS

We consider two forms of transfer. The first asks whether a selector learned for one learner captures the same ordering for another learner. The second asks whether learner-specific selection and adaptive reuse remain useful on real student programs.

G.1 LEARNER-SPECIFIC SELECTOR TRANSFER

To test whether the selection policy is genuinely learner-specific, we compare a selector with the selector fitted from another learner on the same task states. Table 15 reports the matched-versus-swapped selector comparison for DeepSeek Flash and GPT-5.6 Sol.

Table 15: **Learner-specific selector cross-swap.** Each learner supplies one probe per candidate; later repairs evaluate either its matched selector or the other learner's selector. The two selectors disagree on 86.1% of tasks.

Learner	Matched exact	Swapped exact	Δ exact	Δ BProg
DeepSeek Flash	27.2	21.9	$+5.26$	$+7.12$
GPT-5.6 Sol	40.3	35.4	$+4.86$	$+2.34$

The two selectors disagree on most tasks, and each learner obtains a higher point estimate when paired with its matched selector. Together with the multi-model mechanism analysis in Appendix D.3, it indicates that different learners can respond differently to the same counterexample pool.

G.2 Frozen selection on public programs

We next evaluate the frozen selector on C-Pack and IntroClass. Unlike the controlled domains, these tasks use real student programs and benchmark-native test suites. Table 16 reports the three frozen blind comparisons on C-Pack and IntroClass.

Table 16: **Public-program transfer under frozen selection.** Frozen TEACHCE (ours) versus information gain (IG).

Benchmark	Tasks	Exact ours / IG (%)	Progress ours / IG (%)	ΔProgress	95% CI	Result
C-Pack Blind1	80	64.2 / 61.3	56.6 / 50.5	+6.10	$[-0.67, 13.23]$	n.s.
C-Pack Blind2	60	70.7 / 71.7	53.9 / 51.8	+2.17	$[-8.83, 13.67]$	n.s.
IntroClass Blind1	59	49.6 / 50.1	29.6 / 27.9	+1.71	$[-2.28, 6.27]$	n.s.

Each row was frozen before its blind API calls. Point estimates use task means and confidence intervals resample tasks.

The learner-specific selector has positive point estimates on several outcomes, but none of the three pre-specified blind comparisons yields a confidence interval excluding zero. We therefore do not interpret these experiments as evidence that the static selector uniformly outperforms symbolic selection on real programs. Their value is instead to show that the informativeness–teachability mismatch identified in the controlled tasks also arises in executable program repair, while static transfer remains difficult.

G.3 Adaptive repair on public programs

The adaptive setting is more favorable because a failed first repair can be followed by a different counterexample while retaining the best verified candidate. Table 17 compares one-call selection with adaptive two-stage counterexample orderings on the 199 public C programs.

Table 17: **Adaptive rollout reuse on 199 public C programs.** Plausible repairs the shown failure without breaking an originally passing test; Test-correct passes the complete public suite. Values except Calls are percentages.

Policy	Plaus.	Test-corr.	Progress	Reg.	Calls
Max information, one	61.8	60.3	44.9	21.6	1.00
Shortest, one	63.8	59.3	47.4	22.6	1.00
TEACHCE, one	57.3	56.3	42.0	24.1	1.00
Information → TEACHCE	68.3	66.3	54.0	**18.1**	**1.26**
Shortest → TEACHCE	**70.9**	65.8	55.9	18.6	1.28
TEACHCE → shortest	**70.9**	**66.8**	**56.5**	18.6	1.34

Adaptive reuse substantially improves over one-call policies. The strongest learner-specific ordering is competitive with the best alternative while using only modestly more than one model call on average. These results are consistent with the main-paper conclusion: on real programs, the strongest evidence is for adaptive quality–compute trade-offs rather than for a universally superior static counterexample ranking.

H Additional Metrics and Sensitivity Analyses

The main experiments emphasize exact recovery, semantic progress, and catastrophic regression. This section checks whether the conclusions depend on those particular summaries.

H.1 SECONDARY OUTCOMES IN THE PAIRED-BACKBONE COMPARISON

Table 18 reports the secondary outcomes of the paired-backbone comparison, including bounded progress, safe uptake, regression, and log-downside. Bounded progress clips call-level normalized progress to $[-1, 1]$, reducing the influence of rare repairs that increase semantic error by many multiples of the initial error mass.

Table 18: **Secondary outcomes for the paired two-backbone replay.** BProg clips call-level normalized progress to $[-1, 1]$; Safe and Reg are safe-uptake and any-regression rates. All percentage entries average 36 tasks and five repairs per selected CE.

	DeepSeek Flash				GPT-5.6 Sol			
Teacher	BProg	Safe	Reg	LogDown	BProg	Safe	Reg	LogDown
First CE / CEGIS	11.4	37.2	53.3	0.189	9.5	47.2	45.0	0.223
Random CE	9.7	27.2	57.2	0.159	21.6	45.0	42.8	0.173
Shortest CE	12.5	34.4	55.0	0.145	13.5	41.7	48.3	0.176
Max symbolic information	16.5	31.7	49.4	0.202	16.5	38.3	51.1	0.213
Entropy gain	15.8	31.1	50.6	0.206	17.9	37.8	52.8	0.213
Oracle-conditioned elimination	7.7	33.3	53.3	0.309	12.6	36.7	50.0	0.213
Pool diversity	14.1	31.1	55.0	0.164	15.8	40.0	50.6	0.171
Predicted safe uptake	15.3	37.8	43.9	0.184	29.5	46.7	35.6	0.217
Predicted exact recovery	13.7	30.6	55.0	0.167	17.6	38.9	47.8	0.176
Predicted progress	18.6	32.2	52.8	0.089	29.8	46.7	38.9	0.152
Learner-aware elimination	14.7	40.6	43.3	0.203	19.8	46.7	38.9	0.291
TEACHCE (ours)	26.9	40.6	39.4	0.074	31.7	45.0	36.1	0.111

The learner-specific selector remains competitive under these alternative outcomes. In particular, its advantage is not confined to exact recovery: the same ordering generally improves bounded progress and reduces downside relative to symbolic selectors. The full table also makes clear that raw unclipped progress can be dominated by a small number of severe failures, especially in Regex.

H.2 REGEX HEAVY-TAIL SENSITIVITY

Regex is the task family in which normalized progress has the heaviest negative tail. Because a single repair can introduce many more errors than were present initially, we perform a post-hoc distributional sensitivity analysis rather than relying only on the raw mean. Figure 6 visualizes the full task-level progress distribution and its clipped counterpart, showing that the discrepancy is concentrated in the negative tail.

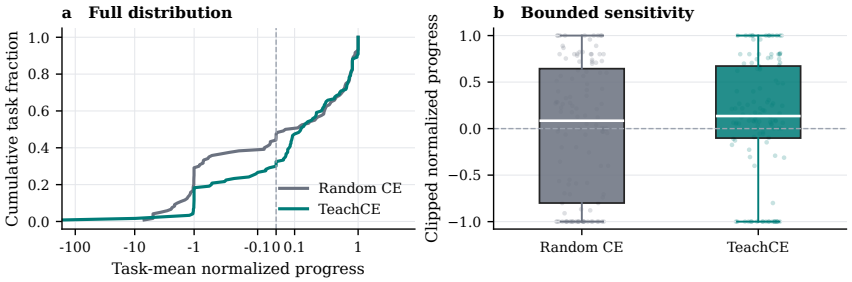


Figure 6: **Regex mean progress is influenced by catastrophic tails.** (a) Task-level normalized progress on a symmetric-log scale. (b) Sensitivity analysis after clipping call-level normalized progress to $[-1, 1]$ before task averaging. The primary endpoint remains the original unclipped progress measure.

Relative to random counterexample selection, TEACHCE improves bounded progress by 13.60 percentage points (95% task-bootstrap CI $[6.91, 21.23]$), reduces the fraction of calls with normalized progress below -1 by 12.33 points ($[6.83, 18.17]$), and increases raw semantic-error reduction by 306.9 errors per task-call ($[126.4, 494.0]$). The corresponding task-median normalized-progress difference is +5.02 points.

These summaries indicate that the disagreement between bounded and unbounded progress is concentrated in a relatively small catastrophic tail. Because this analysis was introduced after inspecting the raw Regex distribution, we treat it as a sensitivity analysis rather than as additional confirmatory evidence.

I LIMITATIONS

Our study focuses on executable synthesis and repair settings where learner outputs can be checked by a verifier. The controlled domains provide exact semantic evaluation, but they do not capture the full complexity of large-scale software systems.

J REPRESENTATIVE REPAIR PROMPTS

The following examples are exported from the same prompt constructors and the first deterministic task–counterexample pair in each controlled benchmark. The bracketed role markers are presentation labels. The system and user messages are sent as separate chat roles. Each user message contains the current executable hypothesis and one labeled counterexample, but not the target hypothesis. Across experimental calls, only the serialized task and counterexample values change. The public C benchmark uses the same response schema but includes the complete current C90 source in the user message. The final retry box is complete: `<MODEL_RESPONSE>` denotes the entire first response inserted verbatim at runtime.

For sequential repair, the same base schema is extended with the labeled counterexamples accumulated so far, eight preservation examples, and up to three rejected revisions with verifier feedback. These fields expose only information produced by prior interaction. The target rule is never included.

Datalog rule repair

```
[SYSTEM]
You repair rules in a controlled Datalog language.
Return exactly one JSON object with keys diagnosis, revised_rule, and change_summary.
The revised_rule must have this grammar:
target(X) :- literal(X), not other_literal(X).
Use only predicates listed in the request. Do not patch a single entity or add constants.

[USER]
{
  "allowed_predicates": [
    "p0",
    "p1",
    "p2",
    "p3",
    "p4",
    "p5",
    "p6",
    "p7",
    "p8"
  ],
  "current_rule": "target(X) :- p3(X), not p5(X).",
  "counterexample": {
    "facts_true": [
      "p0",
      "p3",
      "p6",
      "p7",
      "p8"
    ],
    "all_unlisted_facts_are_false": true
  },
  "current_prediction": true,
  "correct_output": false,
```

Datalog rule repair (continued)

```
  "instruction": "Revise the rule structurally and return JSON only."
}
```

Regular-expression repair

```
[SYSTEM]
You repair regular expressions in a controlled finite language.
Return exactly one JSON object with keys diagnosis, revised_rule, and change_summary.
The revised_rule must be a single Python-compatible regular expression, without slash
     delimiters.
Use only the alphabet listed in the request. Do not use lookarounds or backreferences, and
     do not
patch only the displayed string. Preserve intended behavior while making a structural
     repair.

[USER]
{
  "matching_semantics": "Python re.search over strings up to the stated maximum length",
  "allowed_alphabet": [
    "a",
    "b",
    "c",
    "d"
  ],
  "maximum_evaluation_string_length": 6,
  "current_regex": "^b[abcd]{1,3}$",
  "counterexample": {
    "string": "daaa"
  },
  "current_prediction": false,
  "correct_output": true,
  "instruction": "Revise the regex structurally and return JSON only."
}
```

List-DSL program repair

```
[SYSTEM]
You repair list-transformation programs in a controlled pipeline DSL.
Return exactly one JSON object with keys diagnosis, revised_rule, and change_summary.
The revised_rule must contain one to four operations separated by ` | `.
Allowed operations are identity, reverse, sort_asc, sort_desc, dedup, take:n, drop:n,
filter_eq:v, filter_neq:v, and map_add:k. Use only the stated integer alphabet and bounds.
Do not patch only the displayed input; make a structural program repair.

[USER]
{
  "execution_semantics": "Apply operations from left to right to the whole list",
  "operation_reference": {
    "identity": "leave the list unchanged",
    "reverse": "reverse list order",
    "sort_asc/sort_desc": "sort ascending/descending",
    "dedup": "keep the first occurrence of each value",
    "take:n/drop:n": "keep the first n / remove the first n values",
    "filter_eq:v/filter_neq:v": "keep values equal / not equal to v",
    "map_add:k": "add k modulo alphabet size to every value"
  },
  "allowed_alphabet": [
    0,
    1,
    2,
    3
  ],
  "maximum_input_length": 5,
  "maximum_operations": 4,
  "current_program": "filter_neq:1 | take:1 | drop:1",
  "counterexample": {
    "input_list": [
      0,
      0
    ]
  },
  "current_output": [],
  "correct_output": [
    0
  ],
```

List-DSL program repair (continued)

```
  "instruction": "Revise the program structurally and return JSON only."
}
```

Deterministic format retry (user message)

```
Reformat the content below without changing its proposed rule or diagnosis. Return only
     valid JSON with exactly the string keys diagnosis, revised_rule, and change_summary.

CONTENT:
<MODEL_RESPONSE>
```