WHEN CODE CHANGES, ANSWERS DON'T: BREAKING SEMANTIC INERTIA IN CODE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Code models increasingly reason about programs that change, yet their predictions often remain anchored to what the program did before an edit. We call this failure *semantic inertia*. A small edit may change the output for one input while preserving it for another, so reliable reasoning requires a selective update rather than a blanket reaction to modified code. Across three execution-tuned Qwen3 models, 93% of edited-program errors following a correct original prediction repeat the original output. On a separate frozen panel of 15 models and checkpoints, semantic inertia spans model families and remains measurable even in the strongest hosted models. We introduce the *Semantic Inertia Benchmark* to measure this capability directly. Each group pairs an original and edited program with a *Changed input* that requires Revision and an *Unchanged input* that requires Preservation; *Joint accuracy* requires all four program–input predictions to be correct. We then propose *Counterfactual Regrounding*, which trains on these execution contrasts, emphasizes edited-program answers, and retains correct original-program judgments. Inference remains a single forward pass on one program and input. On 1,000 held-out APPS-derived groups, Counterfactual Regrounding raises Qwen3-4B's Joint accuracy from 9.5% to 16.0%, lowers inertia from 75.4% to 63.5%, and improves Preservation. Its gains grow with Qwen3 model size, transfer to OpenCoder-1.5B, extend to unseen edit families and open-ended generation, and persist on the separate 2,323-group panel. These results establish selective semantic updating as a distinct capability—and execution contrasts as an effective way to train it.

1 INTRODUCTION

Code does not evolve uniformly. The same edit can change a program's output for one input and leave it untouched for another. A model reasoning about evolving code must therefore solve two complementary problems: revise a prediction when execution changes and preserve it when execution does not. Existing execution benchmarks primarily evaluate programs in isolation, while related counterfactual evaluations do not test these obligations jointly (Gu et al., 2024; Jain et al., 2025; Hasanov et al., 2026; Vashishtha et al., 2026). This leaves a central capability unmeasured: whether a model can update its semantic conclusions selectively.

Figure 1 makes the distinction concrete. Replacing `element < limit` with `element <= limit`, where `limit = 3`, changes the upper input's output from 2 to 5. Repeating 2 is therefore a stale prediction. The lower input still returns 3 under both programs, so preserving 3 is correct. Because each program version is queried independently, success requires the model to infer the edit's consequence for the specific input rather than merely notice that the code changed.

We call the failure to make this distinction *semantic inertia*: after a behavior-changing edit, the model remains anchored to an obsolete semantic conclusion and predicts what the program used to output. This is the dominant error pattern in our controlled diagnostics. Across three execution-tuned Qwen3 models, stale errors account for 93% of edited-program errors following a correct original prediction. The separate model-panel track shows that the same failure spans model families.

Semantic inertia reflects a mismatch between pointwise execution training and evolving code. Training on isolated program–input pairs teaches a model to recover an output, but provides little supervision for revising that conclusion after a local edit. Highly similar program versions can therefore

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.

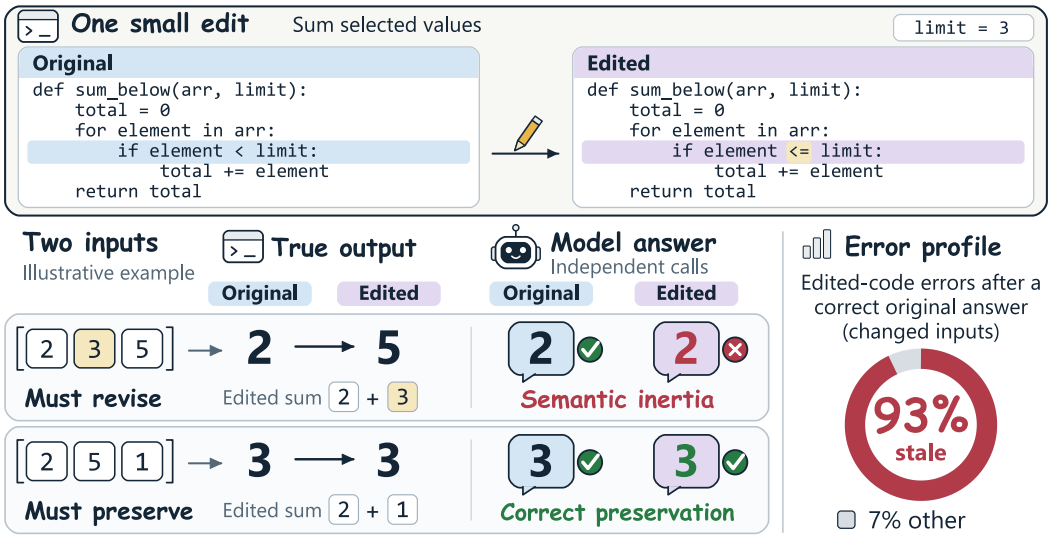

Figure 1: **One edit, two obligations, and semantic inertia.** Top: replacing < with <= includes the boundary value 3 in the sum. Left: the upper row illustrates semantic inertia (the true output changes from 2 to 5, but the model repeats 2); the lower row illustrates correct Preservation because both versions return 3. Illustrative model answers represent independent calls. Right: rounded mean error composition across three execution-tuned Qwen3 models on 576 diagnostic Changed pairs from CRUXEval and LCB-Execution. Among edited-program errors on Changed inputs following a correct original prediction, 93% repeat the original output and 7% are other errors.

activate the same learned execution pattern: the edit registers, but its input-dependent consequence does not reach the answer. Our diagnostics support this account. Edits almost always perturb candidate scores, yet rarely move the relevant margin toward the edited output; on unfamiliar synthetic programs, inertia rises by 29 points at matched original-program accuracy. A model can thus execute a program correctly without representing that execution in a reliably updateable form.

The *Semantic Inertia Benchmark* turns selective updating into a concrete evaluation unit. Each group contains an original program, a single-operator edit, a *Changed input* requiring Revision, and an *Unchanged input* requiring Preservation. The four program–input combinations are presented independently, and *Joint accuracy* requires all four predictions to be correct. We use a 1,000-group held-out APPS method-comparison track and a separate frozen 2,323-group model-panel track.

The benchmark exposes why simply reacting more strongly to edits is not enough. Indiscriminate updating can reduce stale errors yet create spurious switches on Unchanged inputs. The target capability is *selective semantic updating*: revise exactly when execution changes, preserve otherwise, and retain the execution competence already present in the model.

We train this capability with *Counterfactual Regrounding*. An execution contrast pairs the original and edited programs on the same input. Changed inputs teach Revision; Unchanged inputs teach Preservation. Edited-program weighting suppresses stale errors, while retention protects correct original-program judgments. The training signal is contrastive, but the inference interface remains pointwise: one program, one input, and one forward pass, with no execution or paired context.

On held-out APPS, Counterfactual Regrounding raises Qwen3-4B's Joint accuracy from 9.5% to 16.0%, improves Preservation, and reduces inertia from 75.4% to 63.5%. Original-only training stays near the execution-tuned base, while Counterfactual SFT and Revision-only trade higher Revision for lower Preservation. Counterfactual Regrounding instead improves Revision and Preservation jointly. Its gains grow across Qwen3 scales, transfer to OpenCoder-1.5B, and extend to unseen edits, open-ended generation, and the frozen model-panel track.

Our main contributions are:

- **Semantic inertia.** We identify a systematic tendency to retain obsolete semantic conclusions after behavior-changing edits and trace it to failed consequence propagation (§2, §5).
- **Joint evaluation.** The Semantic Inertia Benchmark pairs Changed and Unchanged inputs to measure Revision and Preservation together (§2.3).
- **Counterfactual Regrounding.** Our execution-contrast objective improves selective semantic updating across model scales, an independent backbone, and unseen edit types (§3, §4).

2 Semantic Inertia

We formalize semantic inertia through the two obligations that an edit creates: Revision when execution changes and Preservation when it does not.

2.1 Revision and Preservation under Code Edits

Let P be an original program, P' its edited version, and $E(P, x)$ the output of executing P on input x. Each question contains one complete program, one call, and four candidate outputs. We denote the selected output by $\hat{y}_\theta(P, x)$; a missing or unparseable answer is represented by $\perp$, which differs from every executable output. Questions are independent: the model never sees the other program version or an edit marker.

Changed and Unchanged inputs. For a program pair (P, P'), a *Changed input* x_c and an *Unchanged input* x_u satisfy

$$E(P, x_c) \neq E(P', x_c), \qquad \text{Revision}, \tag{1}$$

$$E(P, x_u) = E(P', x_u), \qquad \text{Preservation}. \tag{2}$$

The same edit can therefore require different prediction relations on different inputs.

2.2 Semantic Inertia and Joint Evaluation

Semantic inertia. On a Changed input, semantic inertia occurs when the model predicts P correctly but repeats the original output for P'. The *inertia rate* is

$$R_{\text{inertia}} = \Pr[\hat{y}_\theta(P', x_c) = E(P, x_c) \mid \hat{y}_\theta(P, x_c) = E(P, x_c)]\,. \tag{3}$$

On a Changed input, a *stale error* occurs when the model selects the original output for P', $\hat{y}_\theta(P', x_c) = E(P, x_c) \neq E(P', x_c)$. The *stale-error share* is the fraction of edited-program errors that have this form, conditioned on a correct original prediction:

$$R_{\text{stale}} = \Pr[\hat{y}_\theta(P', x_c) = E(P, x_c) \mid \hat{y}_\theta(P, x_c) = E(P, x_c),\ \hat{y}_\theta(P', x_c) \neq E(P', x_c)]\,. \tag{4}$$

The stale-error share differs from inertia by excluding correct edited-program predictions from its denominator. On an Unchanged input, the *spurious-switch rate* is the fraction of groups in which the two program versions receive different predictions. An *incorrect switch* is a spurious switch whose edited-program prediction is wrong.

Joint scoring. Each group yields four questions. Let $c_\theta(Q, x) = \mathbf{1}[\hat{y}_\theta(Q, x) = E(Q, x)]$. Revision requires both Changed-input answers to be correct, Preservation both Unchanged-input answers, and Joint all four:

$$A_{\text{Revision}} = \mathbb{E}[c_\theta(P, x_c) c_\theta(P', x_c)]\,, \tag{5}$$

$$A_{\text{Preservation}} = \mathbb{E}[c_\theta(P, x_u) c_\theta(P', x_u)]\,, \tag{6}$$

$$A_{\text{Joint}} = \mathbb{E}[c_\theta(P, x_c) c_\theta(P', x_c) c_\theta(P, x_u) c_\theta(P', x_u)]\,. \tag{7}$$

All four questions share one candidate set but are answered independently. Joint success requires both Revision and Preservation, directly rewarding selective rather than uniform updating.

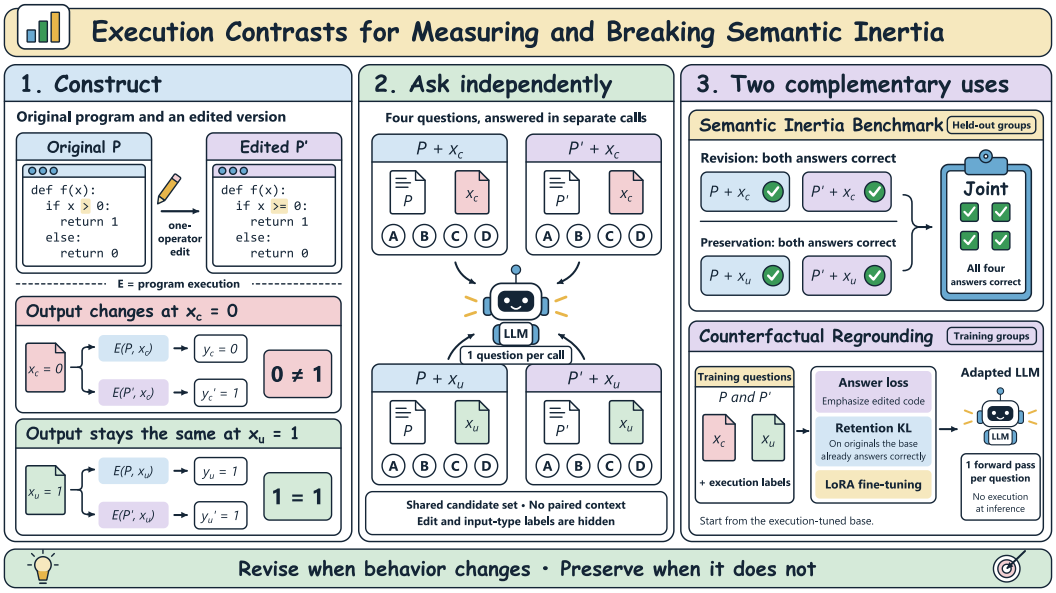

Figure 2: **Execution contrasts for measuring and breaking semantic inertia.** A single edit is evaluated on a Changed input, which requires Revision, and an Unchanged input, which requires Preservation. The four program–input combinations are queried independently. Counterfactual Regrounding trains from the same contrast structure, emphasizing edited-program answers while retaining correct original-program predictions; inference remains a single forward pass.

2.3 The Semantic Inertia Benchmark

Construction. We replace one arithmetic, comparison, Boolean, or bitwise operator in an executable Python program and retain program pairs with deterministic Changed and Unchanged inputs (Figure 2). Executing both versions supplies every label. The four independent questions share candidates containing all true outputs plus deterministic same-type alternatives. This design makes stale repetition directly measurable without revealing the edit or the input's status.

Two tracks. The *method-comparison track* contains 1,000 held-out APPS-derived groups (Hendrycks et al., 2021). The separate *model-panel track* contains 2,323 frozen groups from four sources. The tracks share no source program; Appendix A details their construction.

3 Counterfactual Regrounding

An *execution contrast* pairs the same input with the original and edited programs. Counterfactual Regrounding combines three signals (Figure 2): Changed and Unchanged inputs specify when to revise and preserve, edited-program weighting suppresses stale errors, and retention protects existing execution competence. All methods start from an *execution-tuned base*: a frozen backbone with a low-rank adapter trained on ordinary output prediction. Main experiments use Qwen3 (Yang et al., 2025).

3.1 Learning from Execution Contrasts

Training uses 698 groups from 283 MBPP program clusters (Austin et al., 2021), each with one Changed and one Unchanged input. Presenting the four single-program questions with original and anonymized function names yields 5,584 training questions, all in the same pointwise format used at inference. A *Revision-only* control replaces the Unchanged input with a second Changed input while fixing programs, edits, data volume, and optimization budget. Unchanged supervision is what rules out the shortcut that every edit should change the answer.

3.2 Edited-Program Weighting with Retention

We update rank-2 LoRA adapters (Hu et al., 2022). The answer loss doubles the weight of edited-program questions, while retention keeps predictions close to the execution-tuned base on original-program questions it already answers correctly:

$$\mathcal{L}_{\text{answer}} = -\frac{1}{|B|}\sum_{i\in B} w_i \log p_\theta(t_i \mid z_i), \qquad w_i = \begin{cases} 1, & \text{original program,} \\ 2, & \text{edited program,} \end{cases} \tag{8}$$

$$\mathcal{L}_{\text{retain}} = \frac{1}{\max(1, |C_B|)}\sum_{i\in C_B} \mathrm{KL}(q_0(\cdot \mid z_i) \,\|\, q_\theta(\cdot \mid z_i)), \qquad \mathcal{L} = \mathcal{L}_{\text{answer}} + 3\,\mathcal{L}_{\text{retain}}. \tag{9}$$

Here z_i is a question, t_i its label, and C_B the set of original-program questions answered correctly by the frozen base. The distributions q_0 and q_θ come from the frozen base and adapted model, respectively. Balanced microbatches keep the supervised-loss scale fixed. We reserve *Preservation* for behavior on Unchanged inputs and *retention* for the KL term. Training runs for one epoch with three seeds; Appendix B gives the remaining details. Inference remains one forward pass on one program and input, without execution or paired context.

4 Breaking Semantic Inertia

4.1 Evaluation Setting

Main experiments use Qwen3-4B. Scaling experiments add Qwen3-0.6B and 1.7B, and OpenCoder-1.5B-Instruct provides an independent-backbone replication. APPS is held out for final evaluation; CRUXEval and LCB-Execution guide development and provide 576 diagnostic Changed pairs. Controls include Original-only training, Counterfactual SFT, Revision-only training, and two CodeFort-style objectives adapted to the same data (Zhang et al., 2024). Unless otherwise stated, Counterfactual SFT refers to the development-selected learning rate of 3×10^{-5}; the remaining learning rates in Table 2 expose the Revision–Preservation trade-off. Trained results average three seeds. The prespecified APPS protocol tests a Bonferroni-adjusted mean Joint gain, requires a positive gain for every seed, and limits losses on complementary metrics; Appendix C gives the complete protocol and uncertainty estimates.

4.2 Counterfactual Regrounding Improves Revision and Preservation Jointly

Table 1: Main results (%). Revision and Preservation require both program-version answers to be correct on the Changed and Unchanged inputs, respectively; Joint requires all four. APPS is held out; CRUXEval and LCB-Execution guided development.

	Held-out APPS			Development, Joint	
Method	Revision	Preservation	Joint	CRUXEval	LCB-Execution
Qwen3 backbone	8.1	64.5	5.5	12.8	7.0
Execution-tuned base	11.3	77.7	9.5	19.8	11.6
Original-only	10.9	77.5	9.1	19.6	11.3
Counterfactual SFT	18.8	76.5	15.1	22.5	15.8
Revision-only	**19.3**	76.7	15.3	23.1	15.2
Counterfactual Regrounding	18.6	**79.6**	**16.0**	**24.2**	**16.5**

Table 1 presents the central result. On held-out APPS, Counterfactual Regrounding raises Joint accuracy from 9.5% to 16.0%. The gain is positive for every seed, satisfies the confirmatory protocol, and is matched by the highest Joint accuracy on both development datasets.

The improvement has the signature of selective semantic updating. Revision rises by 7.3 points, Preservation improves by 1.9 points, and inertia falls from 75.4% to 63.5% (Table 2). Counterfactual SFT and Revision-only reach similar Revision but lose Preservation; Counterfactual Regrounding improves both obligations together.

On the 690 groups where both program versions reach the edited node, Counterfactual Regrounding still improves Joint and Preservation. It therefore learns edit consequences rather than edit reachability. Appendix D.3 reports the slice results.

4.3 Execution Contrasts Must Be Selective

The controls separate the two requirements of our account: execution contrasts must supply the missing update signal, and Changed/Unchanged supervision with retention must direct that signal to the right inputs.

Table 2: Controls and ablations on held-out APPS (%). Higher Revision, Preservation, and Joint are better. Inertia measures repetition of the original output within each method's own correct-original cohort; Spurious switch measures prediction changes on Unchanged inputs. Lower is better for both.

Method	Revision	Preservation	Joint	Inertia ↓	Spurious switch ↓
Execution-tuned base	11.3	77.7	9.5	75.4	5.8
Original-only	10.9	77.5	9.1	75.2	7.2
Counterfactual SFT, lr 1.5×10^{-5}	15.4	77.6	13.1	69.8	7.0
Counterfactual SFT, lr 3×10^{-5}	18.8	76.5	15.1	66.5	7.8
Counterfactual SFT, lr 6×10^{-5}	21.5	73.7	16.3	59.7	10.6
CodeFort-style MBA	19.3	76.7	15.5	63.1	8.0
CodeFort-style MBA+ALP	19.0	77.2	15.4	63.9	7.5
Revision-only	19.3	76.7	15.3	63.9	8.9
Counterfactual Regrounding, without retention	18.0	77.9	15.3	64.2	7.4
Counterfactual Regrounding, equal answer weights	17.2	78.9	14.9	66.2	6.5
Counterfactual Regrounding, edited-answer loss only	18.5	78.2	15.7	63.4	6.9
Counterfactual Regrounding	18.6	79.6	16.0	63.5	6.0

Execution contrasts provide the missing semantic-update signal. Original-only training stays near the execution-tuned base, whereas every method trained on execution contrasts raises Revision and Joint (Table 2). The useful signal is therefore not more pointwise execution practice, but supervision over how outputs relate across original and edited programs.

Stronger updating alone is not selective semantic updating. Every execution-contrast objective raises Revision, but the complete Counterfactual Regrounding objective delivers the largest Preservation gain (Figure 3a). Counterfactual SFT at a high learning rate obtains still higher Revision by switching more often: at 6×10^{-5}, Preservation falls from 77.7% to 73.7% and spurious switches rise from 5.8% to 10.6%. The CodeFort-style objectives show the same trade-off. Lower inertia alone is therefore incomplete; the meaningful target is lower inertia together with stronger Preservation.

The complete method achieves selective semantic updating. Ablations clarify the role of each component. Removing Unchanged supervision or retention weakens Preservation or increases spurious switches. Equal answer weights place less emphasis on the edited program, leave more stale errors in place, and reduce Joint. Together, the three components produce the strongest Revision–Preservation balance.

4.4 The Gains Target Stale Errors and Preserve Execution Competence

Training removes stale errors specifically. Counterfactual Regrounding delivers the largest estimated reduction in stale errors among the methods in Figure 3b. Its change in incorrect switches is small and statistically compatible with zero, while Revision-only and Counterfactual SFT increase

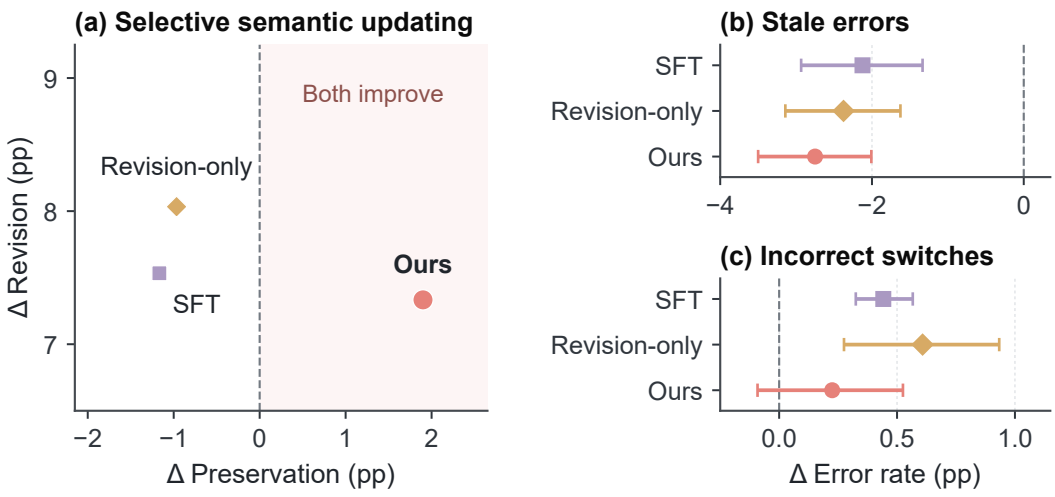

Figure 3: **Selective semantic updating on held-out APPS.** All changes are relative to the execution-tuned base, in percentage points (pp). (a) Revision–Preservation balance on zoomed axes; shading marks improvement in both metrics. Table 2 gives the full controls. (b,c) Percentage-point changes in stale-error and incorrect-switch rates, computed over the 4,000 APPS questions, with paired 95% bootstrap intervals.

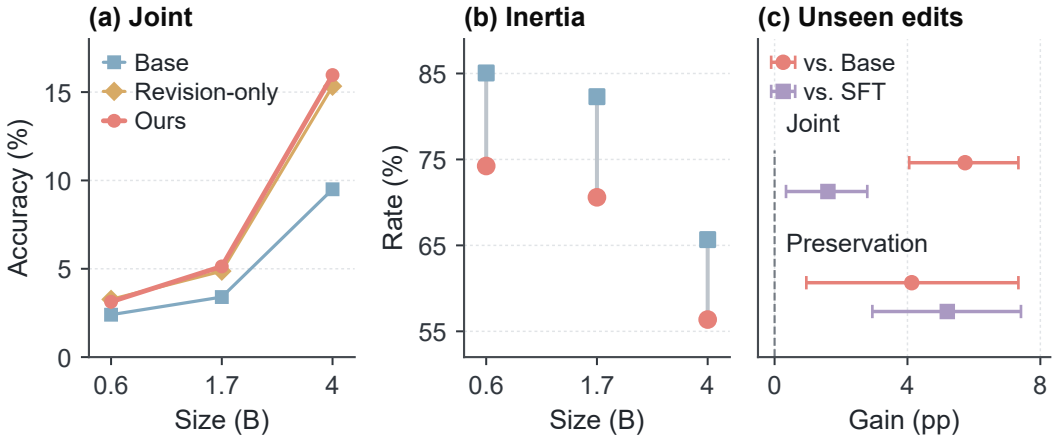

Figure 4: **Generalization across model sizes and edits.** (a) Joint accuracy across Qwen3 sizes on held-out APPS. (b) Inertia on 576 diagnostic Changed pairs for the execution-tuned base (blue squares) and Counterfactual Regrounding (coral circles), each conditioned on its own correct-original cohort. (c) Counterfactual Regrounding's gains over the execution-tuned base and Counterfactual SFT on 872 unseen-edit groups, with paired 95% bootstrap intervals.

them (Figure 3c). The error profile matches the intended behavior: stale errors decline without a corresponding rise in indiscriminate revision. Appendix C details the intervals.

Single-program competence is largely retained. Ordinary output prediction is preserved alongside selective updating: CRUXEval accuracy remains essentially unchanged, while LCB-Execution improves. Counterfactual Regrounding therefore adds updateability while retaining the execution competence of its initialization.

4.5 The Gains Generalize across Models and Edits

Model size. Counterfactual Regrounding improves Joint and Preservation at every Qwen3 size, with larger Joint gains at larger scales (Figure 4a). Larger models begin with lower inertia, yet most

correct-original cases still remain stale after an edit; Counterfactual Regrounding lowers inertia further at every size (Figure 4b). Appendix D reports the full scale results and intervals.

Other model families. Semantic inertia spans model families. Qwen2.5-Coder and DeepSeek-Coder-V2 show high inertia, while GLM-5.3 and Qwen3.8-Max make fewer errors but still produce predominantly stale errors. The training benefit also transfers to OpenCoder-1.5B-Instruct: on held-out APPS, Counterfactual Regrounding raises Joint from 1.0% to 3.4% and Preservation from 63.6% to 66.1%, exceeding Counterfactual SFT on both. Appendix D.10 gives the cross-family diagnostics, and Appendix D.11 reports the replication.

Unseen edit types. Training uses only operator replacements, but the learned behavior transfers to 872 groups with changes to constants, branch structure, or argument references. Counterfactual Regrounding raises Joint from 5.7% to 11.5% and Preservation from 57.5% to 61.6%. It also exceeds Counterfactual SFT on both measures, with paired intervals above zero (Figure 4c). Appendix D.12 reports the edit-family breakdown.

Open-ended generation. Open-ended generation confirms the same pattern on all 576 diagnostic Changed pairs. Counterfactual Regrounding lowers inertia from 52.6% to 43.1%, reduces the stale-error share, and improves edited-program accuracy. Appendix D.14 gives the full results.

4.6 The Frozen Model-Panel Track Confirms the Broader Failure

The frozen model-panel track evaluates 15 models and checkpoints on the same 2,323 groups. Six external models remain below 5% Joint. The four highest-scoring hosted models reach 67.6–84.0% Joint; among groups with four valid answers, most edited-program errors following a correct original prediction are stale. Within the controlled Qwen3-4B checkpoints, Counterfactual Regrounding raises Joint from 7.7% to 10.7% and Preservation from 47.1% to 51.0%, outperforming Counterfactual SFT on both. Appendix E reports the full panel (Table 21).

5 Why Models Fail to Update

Pointwise execution training can produce accurate outputs without making those outputs reliably updateable. Similar program versions activate similar execution patterns; an edit perturbs the model, but its consequence often fails to propagate to the answer. We test this account in three steps: whether edits register, whether unfamiliar compositions amplify inertia, and whether Counterfactual Regrounding restores consequence-sensitive updating.

5.1 Edits Register, but Their Consequences Do Not Propagate

We analyze a representative execution-tuned Qwen3-4B model. Let $s_P(a)$ denote its score for candidate a under program P. For original output y and edited output y', define the relative-margin shift as $\delta = [s_{P'}(y') - s_{P'}(y)] - [s_P(y') - s_P(y)]$. Positive δ means that the edit moves the relative margin toward y', even if y' does not become the top prediction. Among the 720 Changed-input pairs whose predictions remain unchanged, candidate scores move in 99.9% of cases, showing that the edit registers. But the margin moves toward the edited output in only 25.4% (Figure 5a). Across 653 edited-program errors, the original output matches or exceeds the edited output in 97.2% of cases. These score movements identify a failure of consequence propagation: the model responds to the edit without revising the obsolete semantic conclusion.

5.2 Unfamiliar Programs Amplify Semantic Inertia

Figure 5b compares 400 freshly generated pairs with the diagnostic Changed pairs at the same overall original-program accuracy of about 59%. Inertia is 29 points higher on the synthetic programs, with similarly large gaps for arithmetic and comparison edits. Familiar execution patterns can support the original answer without supporting its revision; when those patterns are less reusable, the gap between execution competence and selective semantic updating widens sharply.

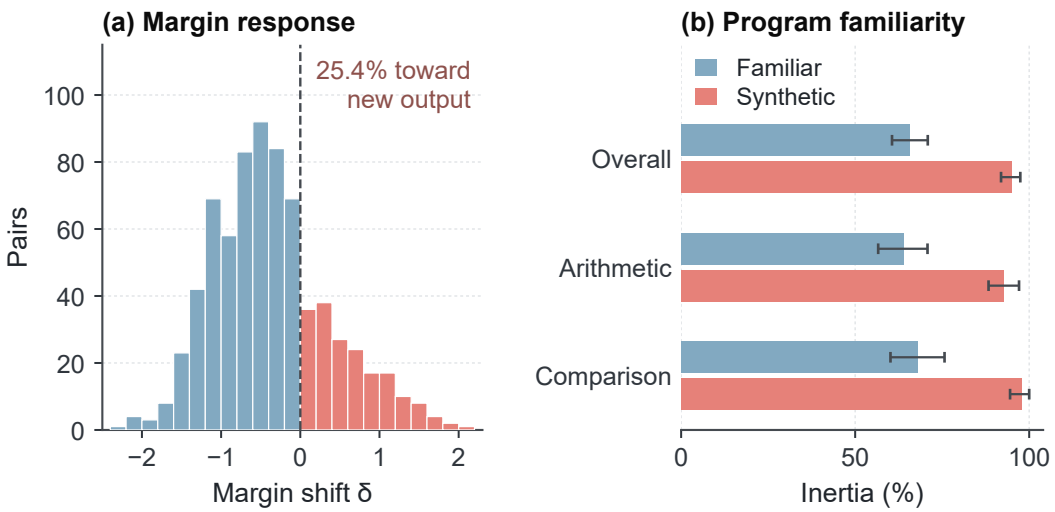

Figure 5: **Diagnostics of semantic inertia.** (a) Relative-margin shifts for the 720 Changed-input pairs with unchanged model predictions, drawn from 872 unseen-edit groups. Positive δ denotes movement toward the edited output; the dashed line marks zero. (b) Inertia on familiar and synthetic programs at similar overall original accuracy, with 95% bootstrap intervals.

5.3 Counterfactual Regrounding Restores Selective Semantic Updating

Hidden-state replacement reveals how Counterfactual Regrounding changes the use of edit information. We replace the original program's operator-position state with its edited-program counterpart and compare the effect with a norm-matched random control. At layer 0, Counterfactual Regrounding shifts the Changed-input margin by 5.09 toward the edited output while limiting Unchanged-input margin loss to 0.25, compared with 0.76 for Revision-only and 0.96 for Counterfactual SFT; only 6.8% of Unchanged predictions flip. At layer 9, it produces both the largest Changed-input shift and the smallest Unchanged-input loss. The transferred edit state is therefore used according to its consequence for the later input—precisely the selective behavior missing from pointwise training. Appendix G provides the full protocol and layerwise results.

6 Related Work

Execution benchmarks evaluate program behavior, while executable counterfactuals intervene on inputs under fixed code (Gu et al., 2024; Jain et al., 2025; Roy et al., 2026; Hasanov et al., 2026; Vashishtha et al., 2026). Code-robustness work studies diverse edits or consistency under semantics-preserving transformations (Aggarwal et al., 2025; Zhang et al., 2024; Ren et al., 2026). Our setting connects these directions: one source-code edit demands Revision on one input and Preservation on another, making selective semantic updating the evaluation target. *Code over Words* uses "semantic inertia" for mutable game rules that conflict with prior knowledge (Xu et al., 2026); we instead study stale outputs after execution-verified source-code edits.

7 Conclusion

Code models often answer an edited program with what it *used to* output. We trace this semantic inertia to failed consequence propagation: the edit registers, but its input-dependent effect does not reliably reach the answer. The Semantic Inertia Benchmark measures selective semantic updating by pairing Revision and Preservation. Counterfactual Regrounding trains this capability with execution contrasts, edited-program weighting, and retention, raising Qwen3-4B's held-out Joint accuracy from 9.5% to 16.0% while lowering inertia and improving Preservation. Its gains across model scales, an independent backbone, unseen edit families, open-ended generation, and the frozen model-panel track establish selective semantic updating as a first-class capability of code models.

AI USE STATEMENT

Generative AI tools were used to help draft portions of the manuscript and to polish its organization and language. All AI-assisted text was reviewed and revised by the authors, who take responsibility for the final manuscript.

REFERENCES

Tushar Aggarwal, Swayam Singh, Abhijeet Awasthi, Aditya Kanade, and Nagarajan Natarajan. NextCoder: Robust adaptation of code LMs to diverse code edits. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 616–638. PMLR, 2025. URL https://proceedings.mlr.press/v267/aggarwal25b.html.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Baichuan AI. Baichuan-M3: Modeling Clinical Inquiry for Reliable Medical Decision-Making, 2026. URL https://arxiv.org/abs/2602.06570.

Baidu. ERNIE 5.0 Technical Report, 2026. URL https://arxiv.org/abs/2602.04705.

DeepSeek-AI. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, 2025. URL https://arxiv.org/abs/2501.12948.

DeepSeek-AI. DeepSeek-V4.1-Flash: Stronger, Faster, More Accessible, 2026. URL https://deepseek.com/news/deepseek-v4-1-flash/.

DeepSeek-AI, Qihao Zhu, Daya Guo, et al. DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence, 2024. URL https://arxiv.org/abs/2406.11931.

Alex Gu, Baptiste Roziere, Hugh James Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida Wang. CRUXEval: A benchmark for code reasoning, understanding and execution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 16568–16621. PMLR, 2024. URL https://proceedings.mlr.press/v235/gu24c.html.

Eshgin Hasanov, Md Mahadi Hassan Sibat, Santu Karmaker, and Aashish Yadavally. The path not taken: Duality in reasoning about program execution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16165–16180. Association for Computational Linguistics, 2026. URL https://aclanthology.org/2026.acl-long.735/.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring Coding Challenge Competence With APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c24cd76e1ce41366a4bbe8a49b02a028-Abstract-round2.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. OpenCoder: The Open Cookbook for Top-Tier Code Large Language Models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025. URL https://aclanthology.org/2025.acl-long.1591/.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/94074dd5a072d28ff75a76dabed43767-Abstract-Conference.html.

Meta AI. Llama 3.2: Revolutionizing Edge AI and Vision with Open, Customizable Models, 2024. URL https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/.

MiniMax. MiniMax M3: Frontier Coding, 1M Context, Native Multimodality, 2026. URL https://www.minimax.io/blog/minimax-m3.

Moonshot AI. Kimi K3: Open Frontier Intelligence, 2026. URL https://www.kimi.com/en/blog/kimi-k3.

Qwen Team. Qwen2.5-Coder Technical Report, 2024. URL https://arxiv.org/abs/2409.12186.

Qwen Team. Qwen3.8-Max, 2026. URL https://www.qwencloud.com/models/qwen3.8-max.

Xiaoning Ren, Qiang Hu, Wei Ma, Chongyang Liu, Yan Li, Yao Zhang, Lingxiao Jiang, Yongqiang Lyu, and Yinxing Xue. ProCURE: Addressing the Programming Concept Understanding Gap for Code Generation in LLMs via Concept-Aware Consistency Learning. In *IJCAI-ECAI 2026*, 2026. URL https://2026.ijcai.org/accepted-papers/?ijtrack=main-track. Main track, paper 7117; preprint arXiv:2508.12620.

Monoshi Kumar Roy, Simin Chen, Benjamin Steenhoek, Jinjun Peng, Gail Kaiser, Baishakhi Ray, and Wei Le. CodeSense: A real-world benchmark and dataset for code semantic reasoning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ehXVDJm0PS.

Aniket Vashishtha, Qirun Dai, Hongyuan Mei, Amit Sharma, Chenhao Tan, and Hao Peng. Executable Counterfactuals: Improving LLMs' Causal Reasoning Through Code. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=Lm46gJA0q8.

Manjie Xu, Isabella Yin, Xinyi Tu, Chi Zhang, and Yixin Zhu. Code over Words: Overcoming Semantic Inertia via Code-Grounded Reasoning, 2026. URL https://arxiv.org/abs/2601.18352.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 Technical Report, 2025. URL https://arxiv.org/abs/2505.09388.

Z.ai. GLM-5.3, 2026. URL https://zcode.z.ai/en/docs/welcome.

Yuhao Zhang, Shiqi Wang, Haifeng Qian, Zijian Wang, Mingyue Shang, Linbo Liu, Sanjay Krishna Gouda, Baishakhi Ray, Murali Krishna Ramanathan, Xiaofei Ma, and Anoop Deoras. CodeFort: Robust training for code generation models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 5262–5277. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.findings-emnlp.303/.

A Data Construction and Evaluation Units

We use the evaluation units defined in Section 2.1. A *group* contains an original program P, its edited version P', one *Changed input* x_c for which $E(P, x_c) \neq E(P', x_c)$, and one *Unchanged input* x_u for which $E(P, x_u) = E(P', x_u)$. Its four program–input combinations are presented as independent *single-program questions*; no question reveals the other program version or the input's status. A *Changed pair* contains only (P, P', x_c) and is used for diagnostics that do not require Preservation scoring.

A.1 Training Sources and Split Boundaries

The execution-tuned bases and Counterfactual Regrounding pool originate from MBPP programs (Austin et al., 2021). The bases are first trained pointwise on ordinary output prediction, matching the isolated-program supervision discussed in Section 3. Preparing this stage starts from 974 source programs and yields 563 training and 208 development source clusters after execution and candidate checks. The resulting training partition contains 3,376 stored questions, organized as 1,688 two-question pairs across 422 groups, and the development partition contains 629 candidate questions; these questions vary concrete calls to ordinary programs. Each execution-tuned base is fixed before counterfactual continuation, so the three continuation seeds measure variation conditional on the same starting checkpoint.

The 4B execution-tuned base starts from `Qwen/Qwen3-4B` (Yang et al., 2025). We train rank-2 query/value LoRA adapters for three epochs with seed 59010, AdamW (betas 0.9/0.95, weight decay 0.01), learning rate 3×10^{-4}, unit pair microbatches, four-step gradient accumulation, gradient checkpointing, and global gradient-norm clipping at 1. The scheduler warms up for 30 optimizer steps and then follows a cosine decay to 10% of the initial learning rate. In every epoch the 1,688 pairs are shuffled and two of the four cyclic option rotations are sampled per pair; four deterministically replayed pair microbatches pad each epoch to 1,692 microbatches, exactly 423 optimizer steps under four-step accumulation. Three epochs therefore give 1,269 optimizer updates and 3,732,526 training tokens. All backbone weights are frozen. The epoch-3 checkpoint is selected by ordinary single-program development accuracy (86.7%). The experiments use Python 3.10.21 and PyTorch 2.5.1+cu121.

The execution-contrast construction reuses only the MBPP training partition and retains edits for which the Counterfactual Regrounding and Revision-only constructions admit the same four-candidate set. This yields 698 groups from 283 training clusters. The MBPP development split contains 238 groups from 102 clusters; presenting every semantic question with both original and anonymized function names produces 1,904 stored questions. The single-program retention check keeps 602 of the 629 candidate questions after excluding 6 with ambiguous outputs, 7 duplicate prompts, 9 execution failures, and 5 length-limit violations. Program clusters and normalized original-code identities are checked for overlap across all partitions, including the pointwise data used for the execution-tuned base. Table 3 summarizes the units.

Table 3: Data units. An evaluation group yields four semantic questions: two programs under a Changed and an Unchanged input. MBPP training and development store each question under two presentation views, yielding eight stored questions per group.

Partition	Groups	Questions	Clusters	Role
MBPP counterfactual training	698	5,584	283	Training
MBPP counterfactual development	238	1,904	102	Development
MBPP single-program development	–	602	–	Single-program check
CRUXEval	172	688	–	Development
LCB-Execution	285	1,140	–	Development
APPS method-comparison track	1,000	4,000	1,000	Held-out test
Model-panel track	2,323	9,292	2,323	Model comparison

A.2 Local Edits and Input-Dependent Labels

A local edit replaces one operator in the program's abstract syntax tree. Replacements cover arithmetic operations, comparisons, Boolean connectives, and bitwise operations, for example addition/subtraction, floor-division/modulo, comparison complements, and conjunction/disjunction; the rest of the program is unchanged. Group membership is determined only by execution on the selected inputs: the same edit may change one output, preserve another, or fail to supply both obligations.

For training, a bounded search seeks a second Changed input and an Unchanged input in addition to the existing Changed call. The outputs of both programs across all three calls must have the same type and admit a common set of four distinct candidates. Counterfactual Regrounding uses one Changed and one Unchanged input, whereas Revision-only uses the two Changed inputs. Selecting both methods from this shared construction controls program identity, edit identity, candidate pool, question count, and optimization steps; their relevant difference is whether training teaches both revision and preservation or revision alone.

The four questions in a group share a candidate set containing every true return value; remaining candidates are deterministic same-type literal perturbations. Numeric aliases and other ambiguities that would let two displayed candidates denote the same answer are excluded. Candidate order is fixed before the primary evaluation, and cyclic reorderings are evaluated only as interface controls (Appendix F).

The frozen model-panel track keeps separate records for units, rendered views, execution-derived labels, and candidate permutations. Its 2,323 groups and 9,292 questions were fixed before model evaluation.

A.3 Constructing the Method-Comparison Track

The held-out method-comparison pool comes from the official APPS training split, not its code-generation test split. It contains 2,926 introductory or interview problems with a nonempty function interface. Before any solution or input field is opened, public identities are filtered for known task URLs, duplicate descriptions, previously used interfaces, and repeated sources, and the list is ordered by a fixed hash. Each problem contributes at most one group, and duplicate normalized functions are removed. No APPS source is used to train the execution-tuned base, Counterfactual Regrounding, or its controls.

For each candidate problem, qualification considers at most three Python solutions exposing a top-level function or a declared method of a `Solution` class with JSON-representable arguments. Standard-library imports, and NumPy when required, are shown to the model; programs depending on files, network, processes, clocks, input reading, printing, or dynamic execution are excluded. Qualification inspects at most 32 published calls in a fixed order and keeps up to eight normally executing ones. Calls are limited to 2,048 characters, and outputs must be finite, round-trip-parseable Python literals of at most 1,024 characters that agree across two executions with different random and hash seeds.

Up to 16 single-operator replacements are examined in a fixed order. An accepted edit must execute stably on the selected calls, preserve output type, and supply at least one Changed and one Unchanged input; each of the four prompts must fit in 2,048 tokens. Every selected program–input combination is re-executed in independent sandbox processes with a 0.4-second CPU limit per call. The prespecified stopping rule keeps the first 1,000 eligible groups; the construction examines 2,528 candidates to reach them.

A.4 Source Overlap and Unchanged-Input Coverage

The CRUXEval derivative excludes exact and identifier-normalized overlap with training programs. Its 244 Changed pairs form 243 source-program clusters, and the 332 LCB-Execution pairs form 76 source-problem clusters. Of these 576 pairs, 119 lack a qualifying Unchanged input under the bounded search; the other 457 form the CRUXEval and LCB-Execution groups. In 246 of them the Unchanged call reaches the edited operation in both programs, and in 211 it does not; both kinds are

kept. A fixed 128-group development subset supports the interface and function-name controls. The two controls use separately rendered evaluation sets, each with its own fixed candidate order.

A code-only check compares the 1,000 APPS groups with 3,912 retained code records using exact Jaccard similarity on five-token shingles, in lexical and identifier-normalized views. No APPS program reaches any of the prespecified thresholds 0.7, 0.8, or 0.9.

B Training Details

B.1 Counterfactual Continuation

Counterfactual Regrounding continues training the existing rank-2 query and value LoRA adapters in all transformer layers; the backbone and native output projection remain frozen, as in the execution-tuned base. The 0.6B, 1.7B, and 4B models have 286,720, 401,408, and 737,280 trainable adapter parameters, respectively. All methods process the complete single-program prompt without truncation. Table 4 lists the shared settings.

Table 4: Optimization settings shared by Ours and the matched controls.

Setting	Value
Adapter scope and rank	Query and value projections, all layers; rank 2
Epochs and optimizer updates	1 epoch; 698 updates
Stored questions and program pairs	5,584 questions; 2,792 pairs
Pair microbatch and accumulation	1 pair; 4 steps
Option orders per pair	2 of 4 cyclic rotations
Learning rate and schedule	3×10^{-5}; 30-update warm-up, cosine to 10%
Optimizer	AdamW, $\beta = (0.9, 0.95)$, weight decay 0.01
Gradient clipping and dropout	Global norm 1; dropout 0
Answer weights (original / edited)	1 / 2
Retention coefficient	3
Training seeds	76016, 76017, 76018
Inference	BF16, batch 8, explicit padding positions

Each balanced microbatch contains both program versions under two sampled cyclic option rotations. The supervised term averages the displayed questions with the 1:2 original-to-edited answer weights in Equation 8. The retention term in Equation 9 is averaged over original-program views that the frozen execution-tuned base answers correctly and requires one frozen-reference forward pass per training context. Thus, *Preservation* remains the behavioral score on Unchanged inputs, whereas *retention* denotes only this KL regularizer on already-correct original-program judgments. The execution-tuned base and adapted models use the native tokenizer interface with thinking disabled.

B.2 Development Selection and Matched Controls

The development comparison crosses the second training input (Changed for Revision-only or Unchanged for Counterfactual Regrounding) with the function-name presentation (original names only, or original plus anonymized names) on the MBPP development split. Adapter interpolation is evaluated at strengths $\{0, .25, .5, .75, 1\}$. A configuration is feasible if it maintains Revision accuracy under both function-name forms and loses at most two points in Preservation or ordinary single-program accuracy. Both input constructions are feasible at full adapter strength; we carry both, with all three seeds and original function names, into the confirmatory APPS evaluation. The Changed/Unchanged construction is Counterfactual Regrounding ("Ours" in tables), and the two-Changed-input construction is Revision-only.

Counterfactual SFT uses the same Changed/Unchanged execution contrasts but removes retention and assigns every question weight 1. Its learning rate is selected from $\{1.5, 3, 6\} \times 10^{-5}$ on the first seed's development results under the same feasibility rule, preferring higher Joint accuracy and then higher single-program accuracy; the selected rate is 3×10^{-5}, and all three rates appear in Table 2. The equal-weight ablation uses answer weights 1.5:1.5 with retention, keeping the total supervised

weight equal to the 1:2 setting. The edited-answer-loss-only ablation uses weights 0:3 with retention. The Original-only control replaces every edited program and its answer with the original counterpart while leaving stored slots, candidates, weights, and optimizer updates unchanged; it therefore adds pointwise execution supervision without an original–edited execution contrast.

B.3 Factor Decomposition

Table 5 directly compares the final Counterfactual Regrounding and Revision-only checkpoints on the 457 complete CRUXEval and LCB-Execution development groups. Replacing Revision-only's second Changed input with an Unchanged input raises Preservation by 5.8 points while keeping Revision comparable. The Joint point estimate also increases. This final-checkpoint comparison isolates the principal role of Unchanged supervision: making the learned update selective rather than simply stronger.

Table 5: Final-checkpoint effect of Unchanged supervision on 457 development groups (Ours minus Revision-only; percentage points, 95% cluster-bootstrap intervals).

Measure	Difference
Δ Revision	-0.8 [-3.2, 1.7]
Δ Preservation	+5.8 [3.1, 8.4]
Δ Joint	+1.2 [-1.2, 3.5]
Δ Spurious switch	-0.9 [-2.7, 1.0]

B.4 Earlier Changed-Pair Methods

Two preliminary methods predate the four-question Changed/Unchanged construction. They train on a 6,672-question mixture containing 1,025 Changed pairs, so they supervise revision but do not jointly teach the complementary preservation obligation. Both use the same adapters, one epoch, learning rate 3×10^{-5}, and seeds 61010–61012. The weighted-supervision method uses original-to-edited answer weights 1:2 and retention coefficient 3, selected from edited-answer weights $\{1, 2\}$ and retention weights $\{0, .3, 1, 3\}$. The pair-loss method uses equal answer weights, retention 0.3, and a pair-matching term with coefficient 0.5: for candidate logits s_P and $s_{P'}$ and true outputs y and y',

$$D = s_P(y) - s_P(y') + s_{P'}(y') - s_{P'}(y), \qquad \mathcal{L}_{\text{pair}} = \log(1 + \exp(-D)). \tag{10}$$

The two preliminary methods perform almost identically on Revision (Appendix I). Counterfactual Regrounding therefore keeps the simpler edited-program weighting and retention, adds the missing Unchanged-input contrast, and evaluates Revision and Preservation jointly.

C Scoring and Statistics

C.1 From Predictions to Group Scores

For locally scored checkpoints, each question is decoded by comparing the four native answer-label logits. We invert the displayed option order to recover the selected candidate value, so semantic predictions—rather than displayed positions—can be compared across program versions and permutations. Revision, Preservation, and Joint are then computed exactly as in Equation 7: both Changed-input answers, both Unchanged-input answers, or all four answers from the same model, seed, and option order must be correct. Missing or unparseable answers are represented by $\perp$ and count as incorrect. For APPS, group indicators are averaged over the three seeds and then over groups; both members of every paired comparison use the same bootstrap resamples. Results on the 576 diagnostic Changed pairs pool all pairs unless explicitly labeled as source-macro averages; resampling remains stratified by source and respects source clusters.

C.2 Inertia, Stale-Error Share, and Intervals

Inertia is computed from candidate values using Equation 3, with each model conditioned on its own correct-original cohort; cohorts must therefore not be compared as if they were identical. Stale-error

share further conditions on the edited-program prediction being wrong, as in Equation 4. On Unchanged inputs, spurious switches compare the two predicted candidate values, not their displayed option letters. On the 576 diagnostic Changed pairs, the three Qwen3 execution-tuned bases average 77.7% inertia and Counterfactual Regrounding averages 67.1%. The reduction of 10.6 points has a 95% interval of [8.1, 12.9]; source clusters are resampled within each source stratum (4,000 replicates), and each execution-tuned base and its adapted seeds share each resample. The 93.4% stale-error share reported in Figure 1 is computed within each model–source cell and then averaged over the six cells formed by three execution-tuned bases and two sources.

For the APPS error decomposition in Figure 3b–c, we count stale errors on Changed inputs and incorrect switches on Unchanged inputs across the 4,000 questions, then pair each continuation seed with its execution-tuned base. Changes for Counterfactual Regrounding, Revision-only, and Counterfactual SFT are, respectively: stale errors -2.8 [-3.5, -2.0], -2.4 [-3.1, -1.6], and -2.1 [-2.9, -1.3]; incorrect switches +0.2 [-0.1, 0.5], +0.6 [0.3, 0.9], and +0.4 [0.3, 0.6]; other errors -0.5 [-1.5, 0.6], -0.5 [-1.8, 0.8], and -0.9 [-2.3, 0.4]. Counterfactual Regrounding reduces the total error rate by 3.0 points [2.1, 3.8]. The familiarity correlations use the same cluster-resampling principle: Spearman $\rho = 0.05$ [-0.06, 0.16] for Qwen3-4B and -0.02 [-0.15, 0.11] for Qwen3-0.6B.

C.3 Primary APPS Tests

The prespecified confirmatory family contains Counterfactual Regrounding and Revision-only at each of the three Qwen3 sizes, giving six comparisons with the execution-tuned base on held-out APPS Joint accuracy. Bonferroni correction assigns each comparison $0.05/6 = 0.0083$; the lower endpoint of the corresponding two-sided interval is therefore the 0.0042 quantile of 100,000 paired bootstrap replicates. The two 4B hypotheses additionally require a mean gain of at least one point, a positive adjusted lower bound, a positive gain for every seed, no decrease in Revision, and declines of at most two points in Preservation and original-program accuracy. Both methods satisfy these criteria (Tables 6 and 7). Comparisons among trained methods and the remaining ablations are secondary analyses.

Table 6: Confirmatory APPS tests: Joint-accuracy gain over the execution-tuned base (percentage points).

Method	Δ Joint	95% interval	Adjusted lower bound
Revision-only	+5.8	[3.9, 7.8]	+3.3
Ours	+6.5	[4.6, 8.3]	+3.9

Table 7: APPS results by training seed (%); Δ Joint is relative to the execution-tuned base (9.5%).

Method	Seed	Revision	Preservation	Joint	Δ Joint
Counterfactual SFT	76016	19.4	76.4	15.6	+6.1
	76017	18.0	76.8	14.4	+4.9
	76018	19.1	76.4	15.4	+5.9
Revision-only	76016	19.9	76.6	15.8	+6.3
	76017	18.5	77.0	14.6	+5.1
	76018	19.6	76.6	15.6	+6.1
Ours	76016	19.2	79.4	16.4	+6.9
	76017	17.8	79.9	15.3	+5.8
	76018	18.9	79.5	16.2	+6.7

D Additional Results

D.1 Full Results by Source

Table 8 extends Table 1 to the failure measures defined in Section 2.2. Among the methods shown, Counterfactual Regrounding has the highest Joint and Preservation accuracy and the lowest inertia

and stale-error share on held-out APPS and both development sources. Revision and inertia improve together, while Preservation and spurious-switch results show that the improvement is selective rather than a uniform tendency to change answers.

Table 8: Results on every source (%). *Spurious switch*: the two program versions receive different predictions on the Unchanged input; *Stale-error share*: fraction of edited-program errors after a correct original prediction that repeat the original output.

Method	Revision	Preservation	Joint	Spurious switch ↓	Inertia ↓	Stale-error share
			APPS (held out)			
Qwen3 backbone	8.1	64.5	5.5	7.8	80.3	93.3
Execution-tuned base	11.3	77.7	9.5	5.8	75.4	90.9
Original-only	10.9	77.5	9.1	7.2	75.2	89.9
Counterfactual SFT	18.8	76.5	15.1	7.8	66.5	92.0
Revision-only	19.3	76.7	15.3	8.9	63.9	91.9
Ours	18.6	79.6	16.0	6.0	63.5	89.6
			CRUXEval			
Qwen3 backbone	20.3	64.5	12.8	7.6	65.1	90.1
Execution-tuned base	22.1	84.3	19.8	5.8	64.1	90.3
Original-only	21.9	84.3	19.6	6.0	64.4	90.4
Counterfactual SFT	26.9	82.0	22.5	6.6	58.9	92.2
Revision-only	27.7	82.2	23.1	6.4	57.3	91.3
Ours	26.7	87.8	24.2	6.0	57.0	88.7
			LCB-Execution			
Qwen3 backbone	12.3	60.0	7.0	7.7	73.3	89.7
Execution-tuned base	15.1	70.9	11.6	6.7	69.2	88.4
Original-only	14.9	71.2	11.3	6.1	69.4	88.2
Counterfactual SFT	23.4	67.6	15.8	7.0	59.8	91.1
Revision-only	22.1	68.3	15.2	7.1	60.6	89.8
Ours	21.4	74.2	16.5	6.0	59.5	86.4

D.2 Accuracy by Program Version

Table 9 splits held-out APPS accuracy by program version and input type. All execution-contrast objectives raise edited-program accuracy on Changed inputs by about ten points relative to the execution-tuned base, which is the marginal most directly suppressed by semantic inertia. Counterfactual Regrounding additionally has the highest accuracy on both program versions for Unchanged inputs. The Joint gain therefore does not come from trading preservation for revision.

Table 9: APPS accuracy by program version and input type (%). *Spurious switch* as in Table 2.

	Changed input		Unchanged input		
Method	Original	Edited	Original	Edited	Spurious switch ↓
Qwen3 backbone	58.5	30.5	67.4	66.9	7.8
Execution-tuned base	66.2	30.8	79.7	80.3	5.8
Original-only	66.4	31.0	80.2	80.6	7.2
Counterfactual SFT	67.8	40.1	80.1	79.3	7.8
Revision-only	63.5	41.5	81.0	80.0	8.9
Ours	63.9	41.0	82.4	81.5	6.0

D.3 Preservation When the Edit Is Reached

For each APPS group, we instrument both program versions and record whether the Unchanged call reaches the edit site in both. This occurs in 690 of 1,000 groups; in the remaining 310, at least one version does not reach the site. The reached subset rules out the shortcut of preserving whenever execution bypasses the edit: the model must propagate the reached edit and determine

that it leaves the final output unchanged. Table 10 shows Joint gains in both subsets. On reached groups, Counterfactual Regrounding raises Joint by 6.5 points [4.2, 8.7] over the execution-tuned base while adding 1.7 Preservation points [1.2, 2.3]. On the remaining groups, it raises Joint by 6.3 points [3.0, 9.7].

Table 10: APPS results by whether the Unchanged call reaches the edit site in both program versions (%); "Not reached" means at least one version does not reach it.

Subset	Method	Revision	Preservation	Joint	Spurious switch ↓
Reached (690)	Execution-tuned base	10.6	77.7	9.1	6.2
	Counterfactual SFT	18.8	77.0	15.1	8.2
	Revision-only	19.1	78.0	15.3	8.5
	Ours	18.2	**79.4**	**15.7**	**6.3**
Not reached (310)	Execution-tuned base	12.9	77.7	10.3	4.8
	Counterfactual SFT	18.8	75.5	15.2	6.9
	Revision-only	**19.9**	74.0	15.4	9.9
	Ours	19.6	**80.0**	**16.7**	5.4

D.4 Controls with Intervals

Table 11 reports paired Joint-accuracy differences between each control and the method it modifies. Original-only training, which removes the original–edited execution contrast, lowers Joint by 6.9 points relative to Counterfactual Regrounding and by 6.2 points relative to Revision-only. This is the largest isolated effect. The Joint differences from removing retention, equalizing answer weights, or keeping only edited-answer loss are smaller and their 95% intervals include zero; their intended roles are instead visible in the complementary Preservation, inertia, and spurious-switch measures in Table 2. Taken together, the controls support the division of labor in the main text: execution contrasts supply the update signal, Unchanged supervision and retention constrain its selectivity, and edited-program weighting helps dislodge stale predictions.

Table 11: APPS controls as paired Joint-accuracy differences from the modified method (percentage points, 95% intervals).

Control	Relative to Ours	Relative to Revision-only
Without retention	−0.6 [−2.1, 0.8]	+0.4 [−1.0, 1.9]
Equal answer weights	−1.0 [−2.6, 0.5]	−0.8 [−2.4, 0.8]
Edited-answer loss only	−0.3 [−1.8, 1.3]	+0.0 [−1.4, 1.5]
Counterfactual SFT	−0.8 [−2.3, 0.7]	−0.2 [−1.6, 1.3]
Original-only	−6.9 [−8.3, −5.5]	−6.2 [−7.7, −4.8]

D.5 CodeFort-Style Objectives

CodeFort (Zhang et al., 2024) trains code-generation models to be robust to semantics-preserving rewrites. We adapt its MBA and MBA+ALP objectives to output selection on the same 698-group data. Each training context pairs its unaugmented view with one semantics-preserving rewrite—consistent renaming, condition negation with exchanged branches, or unreachable empty-branch insertion, tried in a fixed hash order—that reproduces all recorded outputs under isolated execution. MBA applies equal-weight answer cross-entropy to the unaugmented and rewritten views; MBA+ALP adds a KL term between their answer distributions with weight 1. Unlike Counterfactual Regrounding, these objectives enforce invariance to a semantics-preserving rewrite rather than directly supervising how outputs should change across a behavior-changing edit. Both start from the same execution-tuned base with the same adapters, one epoch, and 698 updates; each selects learning rate 6×10^{-5} from $\{1.5, 3, 6\} \times 10^{-5}$ under the rule used for Counterfactual SFT.

Both CodeFort-style objectives improve Joint over the execution-tuned base by 6.0 [4.1, 8.0] and 5.9 [3.9, 7.8] points, respectively, but their higher spurious-switch rates in Table 2 reveal the same revision–preservation trade-off as aggressive Counterfactual SFT. Counterfactual Reground-

ing achieves 2.9 [1.1, 4.5] points higher Preservation than MBA and 2.4 [0.6, 4.2] points higher than MBA+ALP, while also exceeding both in Joint accuracy.

D.6 Single-Program Retention

Table 12 evaluates Revision on the Changed pairs from the 457 complete development groups (172 from CRUXEval and 285 from LCB-Execution), alongside ordinary single-program questions (632 and 425, respectively). The evaluation uses original function names, the earlier diagnostic prompt, canonical option order `[0,1,2,3]`, batch size 8, and native answer-label logit scoring. This section measures *retained execution competence*, not Preservation on Unchanged inputs. Counterfactual Regrounding keeps CRUXEval single-program accuracy within 0.4 points of the execution-tuned base and improves LCB-Execution by 2.7 points, the best result among trained methods on that source.

Table 12: Revision accuracy on 457 development Changed pairs (172 CRUXEval, 285 LCB-Execution) and accuracy on ordinary single-program questions (632 and 425, respectively; %).

Method	Revision accuracy		Single-program accuracy	
	CRUXEval	LCB-Exec.	CRUXEval	LCB-Exec.
Qwen3 backbone	17.4	13.3	60.4	53.2
Execution-tuned base	25.6	14.4	80.1	62.1
Original-only	25.2	16.0	79.9	62.5
Counterfactual SFT	30.6	27.5	78.6	63.2
Revision-only	32.2	25.7	78.4	63.7
Ours, without retention	31.2	25.8	78.7	61.1
Ours, equal answer weights	31.8	24.1	78.7	63.8
Ours, edited-answer loss only	30.6	26.5	79.1	63.3
Ours	31.8	26.7	79.7	64.9

D.7 Smaller Models

Table 13 reports Joint accuracy for Counterfactual Regrounding and Revision-only at each Qwen3 size. Counterfactual Regrounding improves over the corresponding execution-tuned base by 0.7, 1.7, and 6.5 points at 0.6B, 1.7B, and 4B, respectively, and exceeds Revision-only from 1.7B onward. The increasing absolute gain complements Figure 4: scale reduces but does not remove semantic inertia, and execution-contrast training provides an additional benefit at every size.

Table 13: Joint accuracy on held-out APPS across Qwen3 sizes (%); Δ Joint is relative to each size's execution-tuned base.

Size	Method	Joint	Δ Joint [95% interval]	Per-seed Δ Joint
0.6B	Execution-tuned base	2.4	–	–
	Revision-only	3.3	+0.9 [−0.1, 1.8]	+1.0 / +0.7 / +0.9
	Ours	3.1	+0.7 [−0.3, 1.7]	+1.2 / −0.1 / +1.1
1.7B	Execution-tuned base	3.4	–	–
	Revision-only	4.9	+1.5 [0.3, 2.6]	+1.5 / +1.4 / +1.5
	Ours	5.1	+1.7 [0.5, 2.9]	+1.9 / +1.4 / +1.9
4B	Execution-tuned base	9.5	–	–
	Revision-only	15.3	+5.8 [3.9, 7.8]	+6.3 / +5.1 / +6.1
	Ours	16.0	+6.5 [4.6, 8.3]	+6.9 / +5.8 / +6.7

Preservation accuracy rises from 62.0% to 64.0% at 0.6B (+2.0 [1.5, 2.5]), from 62.4% to 64.6% at 1.7B (+2.2 [1.7, 2.8]), and from 77.7% to 79.6% at 4B (+1.9 [1.4, 2.4]).

D.8 Inertia by Edit Type and Location

Table 14 reports inertia on the 576 diagnostic Changed pairs by edit type and code location, averaged across the three Qwen3 sizes. The execution-tuned bases have high inertia for comparison and Boolean edits and for edits inside control flow. Counterfactual Regrounding lowers inertia in every reported slice, including reductions of 19.8 points for Boolean edits and 11.9 points for edits inside control flow. The effect is therefore not confined to one operator family or to straight-line code.

Table 14: Inertia rate by edit type and code location on the 576 diagnostic Changed pairs (%, lower is better; mean over three Qwen3 sizes).

Slice	Pairs	Backbone	Execution-tuned base	Ours
Arithmetic edit	312	77.5	74.5	63.9
Comparison edit	247	82.3	81.5	71.5
Boolean edit	17	85.9	81.9	62.0
Inside control flow	482	81.9	80.1	68.2
Straight-line code	94	69.7	66.3	61.5
Nested block	368	81.5	78.9	66.8
Function top level	208	76.9	75.5	67.5

D.9 Perplexity-Based Familiarity Control

Table 15 bins the 576 diagnostic Changed pairs into equal quartiles of original-program perplexity. Inertia shows no monotone trend for either backbone, matching the near-zero rank correlations in Appendix C. Token-level perplexity is only a coarse proxy for program familiarity, so this control does not establish the mechanism by itself; it shows that the observed inertia is not explained by a simple monotone relationship with original-program likelihood. The synthetic-program comparison in Appendix D.13 provides the stronger composition-level test used in the main analysis.

Table 15: Backbone inertia by original-program perplexity quartile (144 pairs each; Q1 is most familiar). *Cohort*: originally correct pairs.

Model	Quartile	Median perplexity	Cohort	Inertia (%)
Qwen3-4B	Q1	1.72	88	58.0
	Q2	2.61	81	71.6
	Q3	3.68	89	65.2
	Q4	5.95	80	68.8
Qwen3-0.6B	Q1	1.72	69	89.9
	Q2	2.61	61	96.7
	Q3	3.68	71	87.3
	Q4	5.95	59	89.8

D.10 Other Model Families

Table 16 evaluates additional model families on the same 576 diagnostic Changed pairs (Qwen Team, 2024; DeepSeek-AI et al., 2024; Z.ai, 2026; Qwen Team, 2026). Qwen2.5-Coder and DeepSeek-Coder-V2 exhibit high inertia. GLM-5.3 and Qwen3.8-Max answer nearly every original program correctly and have low inertia rates, but the conditional stale-error share remains high: among their few edited-program errors following a correct original answer, most still repeat the original output. This separates the prevalence of errors from their semantic-inertia composition.

D.11 Cross-Family Training Replication

The cross-family replication uses OpenCoder-1.5B-Instruct as the backbone. Its execution-tuned base is trained on the same pointwise single-program data with the same rank-2 adapter protocol. Counterfactual Regrounding then uses the same 698 execution-contrast groups, one Changed and

Table 16: Semantic inertia across model families on the 576 diagnostic Changed pairs (%). *Original*: original-program accuracy; *All*: equal-weight macro-average of the two sources.

Model	Original	Inertia ↓			Stale-error share
		CRUXEval	LCB-Exec.	All	
Qwen2.5-Coder-7B-Instruct	48.1	83.6	90.1	86.8	94.9
DeepSeek-Coder-V2-Lite-Instruct	44.4	78.1	83.1	80.6	93.2
GLM-5.3	99.7	0.8	1.8	1.3	92.9
Qwen3.8-Max	100.0	0.4	0.6	0.5	75.0

one Unchanged input per edit, 1:2 original-to-edited answer weights, retention coefficient 3, the same update budget, and three seeds. Table 17 compares it with Counterfactual SFT and Revision-only. Counterfactual Regrounding achieves the highest Joint and Preservation at all three endpoints. On held-out APPS and development groups with anonymized function names, its Joint gains over Counterfactual SFT are 1.0 point [0.2, 1.7] and 3.4 points [1.0, 6.0], respectively; the corresponding Preservation gains are 5.5 points [3.7, 7.5] and 7.0 points [1.0, 13.0]. This reproduces the selective-updating advantage on an independent backbone rather than only another Qwen3 scale.

Table 17: OpenCoder-1.5B-Instruct replication (%). *Anonymized names*: 128 development groups with anonymized function names.

Endpoint	Method	Revision	Preservation	Joint	Inertia ↓
Development (457)	Backbone	0.7	23.9	0.7	94.2
	Execution-tuned base	3.1	68.9	2.4	85.0
	Counterfactual SFT	7.2	65.7	3.9	71.4
	Revision-only	7.0	67.0	4.4	72.5
	Ours	**8.0**	**71.3**	**6.6**	**70.7**
Anonymized names (128)	Backbone	0.0	23.4	0.0	97.3
	Execution-tuned base	1.6	70.3	0.0	90.2
	Counterfactual SFT	9.1	66.4	4.7	66.7
	Revision-only	10.9	68.0	5.5	**62.2**
	Ours	**11.7**	**73.4**	**8.1**	62.7
APPS (1,000)	Backbone	0.5	20.8	0.0	95.3
	Execution-tuned base	1.7	63.6	1.0	90.7
	Counterfactual SFT	3.9	60.6	2.4	83.8
	Revision-only	3.7	63.2	2.6	85.0
	Ours	**4.3**	**66.1**	**3.4**	83.9

D.12 Unseen Edit Types

The unseen-edit construction attempts 1,144 edits on APPS-derived programs reserved from training and all other evaluation sets and accepts 872 complete four-question groups (76.2%): 644 constant-value, 91 branch-structure, and 137 argument-reference edits. Each accepted edit has both a Changed and an Unchanged input under a shared four-candidate set. Training uses only operator replacements, so none of these three edit families appears in Counterfactual Regrounding or its controls.

Table 18: Generalization to 872 groups with unseen edit types (%).

Method	Revision	Preservation	Joint	Inertia ↓	Spurious switch ↓
Execution-tuned base	9.7	57.5	5.7	73.8	8.1
Counterfactual SFT	**17.0**	56.4	9.9	**55.2**	9.0
Revision-only	16.0	56.8	9.4	57.8	8.8
Ours	16.8	**61.6**	**11.5**	56.3	**7.5**

Counterfactual Regrounding improves Revision by 7.1 points [5.0, 9.1], Preservation by 4.1 points [1.0, 7.3], and Joint by 5.7 points [4.1, 7.3] over the execution-tuned base. Relative to Counterfactual SFT, it gains 5.2 Preservation points [2.9, 7.4] and 1.6 Joint points [0.3, 2.8], while reducing spurious switches by 1.6 points [0.0, 3.1]. Its Joint gain over the execution-tuned base is 5.9 points [4.0, 7.8] on constant-value edits and 7.1 points [2.7, 11.4] on argument-reference edits; Preservation also rises by 8.8 points [0.7, 16.8] on branch-structure edits. These results support transfer of selective semantic updating beyond the operator replacements seen during training. Figure 5a uses the Changed-input questions from these same groups.

Score-movement diagnostic. We use the representative execution-tuned Qwen3-4B checkpoint and map every displayed answer label back to its candidate value before comparing versions. Let $s_P(a)$ be the answer-label logit assigned to candidate a under program P, and let $y = E(P, x_c)$ and $y' = E(P', x_c)$. The relative-margin shift

$$\delta = \big[s_{P'}(y') - s_{P'}(y)\big] - \big[s_P(y') - s_P(y)\big]$$

is positive when the edit moves the edited output ahead of the original output in relative terms, whether or not it becomes the top-scoring candidate. The model gives the same predicted candidate to both versions in 720 of the 872 groups. Candidate-score vectors differ in 719 of those 720 cases (99.9%), but $\delta > 0$ in only 183 (25.4%). Thus, the edit almost always perturbs the model while rarely moving the relevant margin in the required direction. Among all 653 incorrect edited-program predictions, the original output's score matches or exceeds the edited output's score in 635 cases (97.2%). These two checks distinguish failed consequence propagation from simple edit blindness and supply the counts summarized in Figure 5a.

Attribution-gated objective. For the exploratory reverse-replacement analysis only, a follow-up objective adds two consistency terms to the edited-program-weighted answer loss: Jensen–Shannon agreement between original-name and anonymized-name views and a relation term across program versions. Its matched control keeps only the edited-program-weighted answer loss; both use the same training examples, update budget, and three fresh seeds (76022–76024). On the unseen-edit groups, the attribution-gated objective and its control reach Revision of 16.6% (+6.8 [4.8, 8.9]) and 16.3% (+6.5 [4.5, 8.5]), respectively. These checkpoints are not the Counterfactual Regrounding models reported in the main performance tables; they are used only in Appendix G.3 to probe consequence propagation on unseen edits.

D.13 Synthetic-Program Familiarity Test

A seeded generator creates short numeric or Boolean functions in the CRUXEval style. Each function combines random syllable identifiers (excluding keywords, builtins, and frequent CRUXEval names), a randomly parameterized structure (an accumulation loop, conditional chain, or counting loop of three to eight lines), and random constants; its name is fixed to `f`, as in the familiar pairs. Edits, sandboxed execution, and candidate construction reuse the benchmark pipeline. The accepted pool contains 1,161 Changed pairs; 80.2% of edits lie inside control flow, similar to 83.7% for the familiar pairs. Randomly composing identifiers, structures, and constants is designed to reduce reuse of familiar program templates while retaining comparable surface form and execution demands; it is a familiarity intervention, not a direct audit of pretraining membership.

From the accepted pool, we select 400 pairs in fixed hash order (224 arithmetic, 165 comparison, and 11 Boolean edits). The Qwen3-4B backbone's original-program accuracy on this subset closely matches that on the familiar diagnostic pairs (58.8% versus 58.7%), while inertia is 29.2 points higher [23.5, 35.2] (Table 19). Matching original accuracy separates the ability to obtain a semantic conclusion from the ability to update it: unfamiliar compositions preserve the former on average but sharply degrade the latter, supporting the failed-consequence-propagation account in Section 5.

D.14 Open-Ended Generation

To test whether semantic inertia depends on candidate-based evaluation, we evaluate all 576 diagnostic Changed pairs from CRUXEval and LCB-Execution with open-ended generation. Each model sees three fixed few-shot examples disjoint from the evaluation set and generates greedily for at most 96 new tokens. We parse the first output line with `ast.literal_eval` and compare its

Table 19: Qwen3-4B backbone inertia on synthetic and familiar programs (%, 95% bootstrap intervals). *Cohort*: originally correct pairs.

	Synthetic (400 pairs)		Familiar (576 pairs)	
Slice	Cohort	Inertia	Cohort	Inertia
Overall	235	94.9 [91.9, 97.4]	338	65.7 [60.6, 70.9]
Arithmetic	137	92.7 [88.3, 97.1]	185	63.8 [56.6, 70.8]
Comparison	91	97.8 [94.5, 100.0]	144	68.1 [60.1, 75.7]
Boolean	7	100.0	9	66.7

normalized `repr` with the execution result; parse failures count as errors. For Counterfactual SFT, Revision-only, and Counterfactual Regrounding, each reported result averages three seeds.

Table 20: Open-ended generation on the 576 diagnostic Changed pairs (%). Results are equal-weight macro-averages of CRUXEval and LCB-Execution.

Method	Original	Edited	Inertia ↓	Stale-error share ↓	Parse failure ↓
Execution-tuned base	36.6	12.4	52.6	61.7	1.7
Counterfactual SFT	36.3	15.8	52.0	64.5	2.3
Revision-only	36.1	16.9	51.0	62.5	1.6
Ours	**38.7**	**17.3**	**43.1**	**54.7**	1.6

Under open-ended generation, the execution-tuned base still repeats the original output on 52.6% of its own correct-original cohort; 61.7% of its edited-program errors within that cohort are stale. Counterfactual Regrounding lowers inertia to 43.1% and stale-error share to 54.7%, while raising edited-program accuracy from 12.4% to 17.3%. It also outperforms Counterfactual SFT and Revision-only on both failure measures. Semantic inertia and the benefit of regrounding therefore do not depend on the multiple-choice interface.

E The Model-Panel Track

E.1 Construction and Freezing

The model-panel track applies the same group construction and Joint scoring (Sections 2.3 and 2.2) to 2,323 groups (9,292 questions) from four sources: 2,098 APPS-derived, 212 from CodeContests, 11 from Pallets Click, and 2 from pytest-dev/pluggy. No source program appears in both this track and the method-comparison track. Each source problem contributes at most one group; accepted program–input calls are re-executed in independent sandbox processes, and candidate order is independently permuted for each question. The track was frozen before any panel model was evaluated. It covers four controlled Qwen3-4B checkpoints and eleven external models (four open-weight and seven hosted). Locally run models use their 8,192-token input limit, while hosted models share a common input cap.

The four external open-weight models use official checkpoints from OpenCoder, Llama 3.2, DeepSeek-Coder-V2, and Qwen3 (Huang et al., 2025; Meta AI, 2024; DeepSeek-AI et al., 2024; Yang et al., 2025). The seven hosted models use the provider/model IDs `zhipu/GLM-5.3`, `deepseek/DeepSeek-V4.1-Flash`, `baidu/ERNIE-5.0-Thinking-Preview`, `moonshot/Kimi-K3`, `minimax/MiniMax-M3`, `baichuan/Baichuan-M3`, and `deepseek/DeepSeek-R1` (Z.ai, 2026; DeepSeek-AI, 2026; Baidu, 2026; Moonshot AI, 2026; MiniMax, 2026; Baichuan AI, 2026; DeepSeek-AI, 2025). They receive the same single-program multiple-choice prompt through an OpenAI-compatible gateway with a 32,768-character input cap. Where first-token probabilities for A/B/C/D are available, we select the highest-probability letter; otherwise, we use the first emitted answer letter. Accuracy metrics use all 2,323 groups and count invalid answers as incorrect. Inertia and stale-error share first restrict to groups with four valid answers and then condition on a correct original-program response.

Table 21: Model-panel results on the same 2,323 frozen groups (%). Accuracy metrics count invalid answers as incorrect. Conditional metrics use each model's four-valid group set. Revision and Preservation require both program-version answers to be correct on Changed and Unchanged inputs; Joint requires all four. Lower Inertia and Stale-error share are better.

Model	Revision ↑	Preservation ↑	Joint ↑	Inertia ↓	Stale-error share ↓
Controlled Qwen3-4B checkpoints					
Qwen3-4B backbone	9.8	30.4	4.0	54.7	69.9
Execution-tuned base	14.7	47.1	7.7	56.4	80.0
Counterfactual SFT	18.8	43.7	8.7	46.8	78.0
Ours	19.4	51.0	10.7	45.6	77.1
Open-weight models					
OpenCoder-1.5B-Instruct	6.5	5.6	0.5	21.6	29.0
Llama-3.2-3B-Instruct	8.5	9.7	0.9	39.2	48.2
DeepSeek-Coder-V2-Lite-Instruct	7.7	11.8	1.2	35.5	43.2
Qwen3-4B-Instruct-2507	11.4	30.3	4.2	54.5	71.8
Hosted models					
Baichuan-M3	6.5	7.0	0.4	23.0	31.3
MiniMax-M3	13.1	26.5	4.0	51.9	70.9
DeepSeek-R1	64.6	69.3	52.9	5.6	71.2
Kimi-K3	87.0	93.2	84.0	4.8	74.1
GLM-5.3	85.0	88.8	78.8	2.3	64.3
DeepSeek-V4.1-Flash	86.0	90.3	82.0	1.8	76.1
ERNIE-5.0-Thinking	75.6	78.7	67.6	1.7	93.1

E.2 Content-Level Scoring

Because candidates are permuted independently for each question, displayed positions cannot be compared across program versions. A selected position $\hat{p}$ corresponds to canonical candidate $\mathrm{order}[\hat{p}]$; all semantic-inertia metrics use this candidate identity, and we verified the mapping for all 9,292 label rows. Table 22 shows that position bias does not explain repetition of the original *output*: OpenCoder selects the second displayed position on 99.3% of questions and Baichuan-M3 the fourth on 79.7%, yet their edited predictions repeat the original output on only 24.3% and 24.8% of scored groups, respectively. For every other checkpoint, the most frequent displayed position accounts for only 25.2–25.7% of answers.

E.3 Coverage

Table 24 reports valid-answer coverage for every model. In the primary accuracy metrics, groups with any invalid answer already count as Joint failures; under this conservative convention, the five strongest hosted models reach 52.9–84.0% Joint and Counterfactual Regrounding remains ahead of the execution-tuned base (10.7% versus 7.7%). Within each model's groups with four valid answers, stale errors account for 79 of 111, 103 of 139, 45 of 70, 35 of 46, and 27 of 29 edited-program errors following a correct original prediction, respectively, in their order in Table 21. Thus, high overall accuracy lowers the number of failures but does not eliminate the stale-error pattern among the failures that remain.

As a coverage control, Table 23 compares all 15 models and checkpoints on the same 885 groups with four valid answers from every model. This complete-case subset is not the primary panel estimate because requiring every model to answer can change its difficulty distribution. Within it, Counterfactual Regrounding reaches 18.4% Joint, above the execution-tuned base (14.1%), Counterfactual SFT (16.3%), every external open-weight model (at most 8.4%), and the two lower-performing hosted models (at most 8.8%).

Table 22: Position preferences on the panel (%). *Top position*: share of the most frequent displayed position; *Same position*: edited and original answers at the same displayed position; *Repeat*: edited answer equals the original output, over all scored groups.

Model	Top position	Position	Same position	Repeat
Qwen3-4B backbone	25.7	2nd	24.8	31.5
Execution-tuned base	25.3	4th	23.4	34.2
Counterfactual SFT	25.4	4th	24.4	27.5
Ours	25.7	4th	25.3	27.0
OpenCoder-1.5B-Instruct	99.3	2nd	98.5	24.3
Llama-3.2-3B-Instruct	25.6	4th	25.3	27.7
DeepSeek-Coder-V2-Lite-Instruct	25.2	4th	24.0	24.8
Qwen3-4B-Instruct-2507	25.4	3rd	25.4	32.5
Baichuan-M3	79.7	4th	64.0	24.8
MiniMax-M3	25.4	3rd	24.4	32.7
DeepSeek-R1	25.6	4th	25.9	6.3
Kimi-K3	25.7	4th	25.2	5.3
GLM-5.3	25.5	4th	26.3	2.7
DeepSeek-V4.1-Flash	25.5	4th	25.8	2.1
ERNIE-5.0-Thinking	25.6	4th	26.4	1.8

Table 23: Model-panel results on the 885 groups completed by every model (%).

Model	Revision	Preservation	Joint	Inertia ↓
Qwen3-4B backbone	17.5	45.1	7.7	52.3
Execution-tuned base	25.4	60.7	14.1	51.5
Counterfactual SFT	32.2	56.7	16.3	**39.5**
Ours	31.4	**64.1**	**18.4**	43.6
Qwen3-4B-Instruct-2507	19.4	45.3	8.4	51.2
DeepSeek-Coder-V2-Lite-Instruct	14.2	18.2	2.4	34.2
Llama-3.2-3B-Instruct	15.3	15.6	1.9	39.5
OpenCoder-1.5B-Instruct	7.1	6.4	0.6	19.1
MiniMax-M3	23.1	38.3	8.8	44.9
Baichuan-M3	8.6	9.4	0.8	23.1
DeepSeek-R1	91.0	94.8	87.5	3.9
Kimi-K3	96.9	98.6	95.9	1.9
GLM-5.3	97.2	98.4	95.8	1.8
DeepSeek-V4.1-Flash	99.2	99.7	99.0	0.5
ERNIE-5.0-Thinking	99.0	99.7	98.6	0.9

E.4 Results by Source

Table 25 reports the two larger sources, APPS-derived and CodeContests; Pallets Click and pytest-dev/pluggy contribute only 8–11 and 1–2 scored groups per model, respectively. Across both sources, Counterfactual Regrounding has the highest Joint among the controlled checkpoints and lowers inertia relative to the execution-tuned base, from 56.0% to 46.2% on APPS-derived groups and from 60.2% to 42.0% on CodeContests.

F Prompts and Interface Controls

F.1 The Primary Prompt

The system message is:

```
You answer multiple-choice questions accurately and concisely.
```

Table 24: Panel coverage (9,292 questions, 2,323 groups). *Length*, *Stop*, *Conn.*: API failures from output-budget exhaustion, a stop without an answer letter, and dropped connections.

Model	Valid	Skipped	Length	Stop	Conn.	Scored groups
Controlled Qwen3-4B (each)	9,132	160	0	0	0	2,274
Qwen3-4B-Instruct-2507	9,132	160	0	0	0	2,274
OpenCoder-1.5B-Instruct	9,140	152	0	0	0	2,275
Llama-3.2-3B-Instruct	9,142	150	0	0	0	2,276
DeepSeek-Coder-V2-Lite-Instruct	9,098	194	0	0	0	2,264
Baichuan-M3	8,907	134	248	3	0	2,083
MiniMax-M3	9,155	134	0	3	0	2,278
DeepSeek-R1	7,792	134	1,310	48	8	1,531
Kimi-K3	9,157	134	1	0	0	2,280
GLM-5.3	8,838	134	320	0	0	2,041
DeepSeek-V4.1-Flash	8,746	134	412	0	0	1,996
ERNIE-5.0-Thinking	7,978	134	1,180	0	0	1,619

Table 25: Panel results by source (%). Both metrics use each model's groups with four valid answers within each source; Inertia additionally conditions on a correct original-program answer.

	APPS-derived		CodeContests	
Model	Joint	Inertia ↓	Joint	Inertia ↓
Qwen3-4B backbone	4.0	55.4	4.8	48.4
Execution-tuned base	7.7	56.0	10.0	60.2
Counterfactual SFT	8.6	47.3	11.5	42.6
Ours	10.6	46.2	13.4	42.0
OpenCoder-1.5B-Instruct	0.5	21.9	0.0	19.4
Llama-3.2-3B-Instruct	0.8	39.0	1.4	40.4
DeepSeek-Coder-V2-Lite-Instruct	1.3	35.5	1.0	35.7
Qwen3-4B-Instruct-2507	4.4	54.5	2.9	55.7
Baichuan-M3	0.4	23.4	1.1	20.0
MiniMax-M3	4.1	51.0	3.3	60.4
DeepSeek-R1	80.3	5.7	79.7	4.7
Kimi-K3	85.6	4.7	85.2	5.6
GLM-5.3	89.6	2.3	90.4	2.2
DeepSeek-V4.1-Flash	95.6	1.8	94.1	2.2
ERNIE-5.0-Thinking	96.9	1.7	97.3	1.4

The user message is rendered as follows, with the complete program, call, and literal candidate values substituted:

```
Choose the correct answer. Respond with only one of: A, B, C, D.

Question: Program:
{code}
Input:
{call}
Output:

A. {candidate_1}
B. {candidate_2}
C. {candidate_3}
D. {candidate_4}

Answer:
```

Each model's native chat template is applied. The primary multiple-choice prompt adds no examples, chain-of-thought requests, edit markers, or paired-program context. Consequently, the inference interface is the ordinary single-program setting described in the main text: one program, one call, and one forward pass. Locally scored checkpoints use thinking-disabled templates and candidate logits at the answer position; hosted models receive the same question text, with answers selected from first-token probabilities over A/B/C/D when available and otherwise from the first emitted answer letter (Appendix E).

F.2 Alternative Instructions

An explicit instruction tests whether stronger wording alone resolves semantic inertia by asking the model to follow the program exactly as written. Its system message is `You predict the return values of Python programs.`, and its user message begins:

```
Determine the Python value returned by the call using the program
    exactly as written. Follow the actual operators, comparisons, and
    control flow. Do not repair the program or substitute what its name
     or comments suggest it should do. Choose the returned value, not
    any printed messages.
```

It then presents fenced `Program:` and `Call:` blocks and a `Candidate return values:` list, ending with `Which candidate is the return value of the call? Reply with exactly one option label (A, B, C, D) and no explanation.` The input-first variant swaps the program and call blocks. Training uses the primary prompt throughout; alternative formats are evaluation-only interface shifts. On the MBPP development split, the execution-tuned base reaches 18.8%, 21.3%, and 16.3% Joint under the primary, code-first, and input-first prompts, respectively, showing that instruction order can change absolute performance but does not itself supply the missing execution-contrast supervision.

Table 26 re-scores the frozen Counterfactual Regrounding and Counterfactual SFT checkpoints under three instruction formats (238 development groups, three option rotations, both naming views, three seeds). Counterfactual Regrounding leads under both explicit prompts by 2.1 [0.7, 3.5] and 2.9 [1.6, 4.3] Joint points, respectively, while the primary-prompt scores are comparable. The learned selective-updating behavior therefore survives prompt rewording without having been trained on those formats.

Table 26: Joint accuracy on the MBPP development split under three instructions (%). Δ: Ours minus Counterfactual SFT, with 95% cluster-bootstrap intervals.

Instruction	Ours	Counterfactual SFT	Δ
Primary	26.8	27.7	−0.8 [−2.3, 0.6]
Explicit, code first	28.5	26.4	+2.1 [0.7, 3.5]
Explicit, input first	23.2	20.2	+2.9 [1.6, 4.3]

F.3 Answer Labels, Function Names, and Candidate Order

Table 27 tests five interface variants on the same 128 development groups: primary A–D labels, numeric labels, W–Z labels, and code-first and input-first instructions. Counterfactual Regrounding has the highest Joint accuracy on four of five variants and exceeds the execution-tuned base on all five. Under the input-first instruction, it reaches 18.5% Joint, 6.0 points above the execution-tuned base (12.5%).

Table 28 evaluates the same 128 groups after replacing the entry function name with an anonymized identifier; candidates, labels, and executed outputs remain unchanged within this control. Counterfactual Regrounding gains 5.2 Joint points over the execution-tuned base with original function names and 4.4 with anonymized names, preserving the benefit when lexical function-name cues are removed. The naming and interface analyses were rendered and run separately, with a fixed candidate order inside each analysis; their absolute scores therefore need not coincide. Table 29 evaluates

Table 27: Joint accuracy under different answer interfaces on 128 development groups (%).

Method	Primary	Numeric	W–Z	Code-first	Input-first
Execution-tuned base	18.8	15.6	9.4	18.0	12.5
Revision-only	21.6	15.4	5.2	22.4	16.1
Ours	24.2	16.1	10.7	21.6	18.5

three fixed candidate orders on the 457 complete CRUXEval and LCB-Execution groups; Counterfactual Regrounding has the highest Joint and Preservation under every order. These controls make a fixed answer label, candidate position, or function name an implausible explanation for the main gain.

Table 28: Function-name robustness on 128 development groups (%).

Method	Function names	Revision	Preservation	Joint
Execution-tuned base	Original	20.3	75.8	18.8
	Anonymous	24.2	69.5	17.2
Revision-only	Original	25.5	73.4	20.8
	Anonymous	28.9	71.9	22.4
Ours	Original	27.1	78.9	24.0
	Anonymous	25.0	76.6	21.6

Table 29: Candidate-order robustness on the 457 CRUXEval and LCB-Execution groups (%). Values are equal-weight macro-averages of the two sources, averaged over seeds for trained methods.

Order	Method	Revision	Preservation	Joint
1	Execution-tuned base	18.6	77.6	15.7
	Revision-only	24.9	75.2	19.1
	Ours	24.1	81.0	20.4
2	Execution-tuned base	19.2	77.1	16.1
	Revision-only	25.5	74.7	19.5
	Ours	24.7	80.4	20.9
3	Execution-tuned base	18.1	76.9	15.0
	Revision-only	24.4	74.6	18.5
	Ours	23.6	80.3	19.8

G Internal Interventions

G.1 Restoring Trained Parameters

For each of the three Counterfactual Regrounding and Revision-only seeds, we restore one group of trained adapter parameters to its execution-tuned base values and re-evaluate the 128-group subset. We restore query projections, value projections, or one of five 12-layer sets: early (layers 1–12), middle (13–24), late (25–36), interleaved, and a fixed random set. Restoring all adapters exactly reproduces the execution-tuned base's candidate scores for all six checkpoints. In both methods, the largest Revision loss follows restoration of the middle third (7.0 points), followed by the value projections (4.7 points); restoring either the early or late third costs only 0.8 points (Table 30). This intervention localizes where the learned revision benefit is most causally sensitive; it does not by itself distinguish selective from indiscriminate updating, which the operator-state study tests next.

G.2 Operator-State Replacement

For the hidden-state study, we evaluate 192 fixed groups at layers 0, 9, 18, and 27 for the execution-tuned base and three seeds each of Counterfactual SFT, Revision-only, and Counterfactual Re-

Table 30: Accuracy change after restoring adapter parameters to execution-tuned base values (percentage points, 95% cluster-bootstrap intervals; 128 groups).

Method	Restored parameters	Δ Revision	Δ Joint
Revision-only	Query projections	−3.1 [−4.9, −1.6]	−2.3 [−4.9, 0.0]
	Value projections	−4.7 [−6.8, −2.9]	−3.1 [−5.7, −0.5]
	Early third	−0.8 [−1.8, 0.0]	0.0 [−2.6, 2.6]
	Middle third	−7.0 [−9.9, −4.4]	−3.9 [−6.8, −1.0]
	Late third	−0.8 [−1.8, 0.0]	−0.8 [−3.6, 1.8]
	Interleaved third	−3.1 [−4.9, −1.6]	−1.6 [−3.9, 0.8]
	Random third	−2.3 [−3.9, −1.0]	−1.6 [−3.9, 0.8]
Ours	Query projections	−2.9 [−4.4, −1.3]	−2.3 [−4.4, −0.3]
	Value projections	−4.7 [−7.0, −2.6]	−3.1 [−5.7, −0.8]
	Early third	−0.8 [−1.8, 0.0]	0.0 [−2.1, 2.1]
	Middle third	−7.0 [−9.6, −4.7]	−4.2 [−6.8, −1.6]
	Late third	−0.8 [−1.8, 0.0]	−0.8 [−2.6, 1.3]
	Interleaved third	−2.9 [−4.7, −1.3]	−1.6 [−3.6, 0.5]
	Random third	−2.1 [−3.6, −0.8]	−1.6 [−3.6, 0.8]

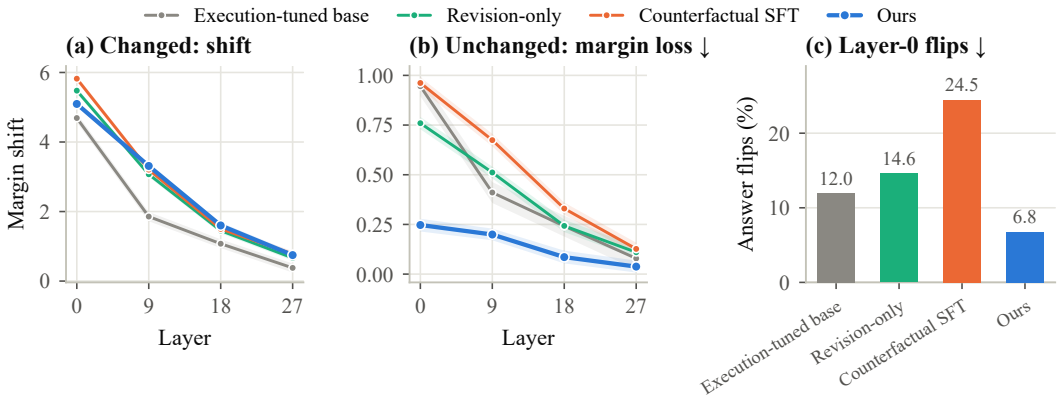

Figure 6: Operator-state replacement on 192 groups; trained methods use three seeds each (layers are zero-based). (a) Shift in the answer margin toward the edited output on Changed inputs. (b) Loss of correct-answer margin on Unchanged inputs (↓). (c) Share of Unchanged-input predictions flipped by a layer-0 replacement (↓). Bands show 95% intervals.

grounding. On each Changed and Unchanged input, we replace the original program's operator-position hidden state with the corresponding state from the edited program. Controls use an identity replacement, an equal-norm random perturbation, and the same state difference inserted at another position. This yields 15,360 group–layer–input records; identity replacement reproduces the original candidate scores in every record.

Let s^{id} and s^{op} be candidate scores under identity and operator-state replacement. On a Changed input with original output y and edited output y', the answer-margin shift is

$$R = \left[s^{\mathrm{op}}(y') - s^{\mathrm{op}}(y)\right] - \left[s^{\mathrm{id}}(y') - s^{\mathrm{id}}(y)\right],$$

so positive R means stronger support for the edited output. On an Unchanged input with correct output y, define $m^r(y) = s^r(y) - \max_{a \neq y} s^r(a)$ and the correct-answer margin loss as $L = m^{\mathrm{id}}(y) - m^{\mathrm{op}}(y)$, so positive L is undesirable. The reported net margins subtract the corresponding R or L induced by the equal-norm random control.

Figure 6 and Table 31 show that replacing early operator states shifts the Changed-input answer margin toward the edited output beyond the equal-norm random control. Counterfactual SFT produces the largest raw layer-0 Changed shift, but also the largest Unchanged disruption. Counterfactual Regrounding instead has the strongest selectivity: at layer 0 it maintains a net Changed shift of 5.09 while limiting Unchanged correct-answer margin loss to 0.25, versus 0.76 for Revision-only

and 0.96 for Counterfactual SFT, and flips only 6.8% of Unchanged predictions. At layer 9 it has both the largest Changed shift (3.31) and the smallest Unchanged loss (0.20). This pattern holds for all three seeds. Because the program precedes the call, the transferred operator state must interact with the later input before the answer is produced; the Changed/Unchanged divergence is therefore consistent with consequence-sensitive propagation rather than mere amplification of an edit signal.

Table 31: Operator-state replacement on 192 groups. *Changed* is the answer-margin shift toward the edited output and *Unchanged* is the correct-answer margin loss, both after subtracting the equal-norm random control. *Flip* is the percentage of Unchanged-input predictions changed by replacement. Trained methods are averaged over three seeds.

Method	Layer	Changed	Unchanged ↓	Flip (%) ↓
Execution-tuned base	0	4.690	0.946	12.0
	9	1.856	0.411	8.3
	18	1.075	0.242	8.3
	27	0.379	0.079	7.3
Counterfactual SFT	0	5.822	0.962	24.5
	9	3.205	0.674	19.8
	18	1.502	0.330	17.9
	27	0.778	0.127	19.1
Revision-only	0	5.477	0.759	14.6
	9	3.072	0.511	16.0
	18	1.451	0.243	13.4
	27	0.666	0.110	12.2
Ours	0	5.095	**0.247**	**6.8**
	9	**3.311**	**0.199**	**3.6**
	18	**1.600**	**0.085**	**3.8**
	27	0.748	**0.038**	**4.2**

G.3 Reverse Replacement on Unseen Edits

To test consequence propagation on edit families absent from training, we select 192 of the 872 unseen-edit groups, 64 in each stratum defined by the execution-tuned base's behavior on the Changed input: *too weak* (incorrect edited-program answer despite a relative-margin shift toward the edited output), *wrong direction* (incorrect answer with a shift that does not favor the edited output), and *success* (correct edited-program answer). The intervention runs in reverse: the edited program's state at the changed token is replaced by the original program's state, at four depths and two positions, with the same controls. The execution-tuned base and each of three exploratory attribution-gated seeds (Appendix D.12) contribute 5,376 records; every identity replacement reproduces the edited-program candidate scores.

Table 32: Layer-0 reverse replacement on 192 unseen-edit groups: answer-margin shift beyond the equal-norm random control (95% intervals). Gated denotes the attribution-gated objective.

Checkpoint	Too weak	Wrong direction	Success
Execution-tuned base	−2.64 [−3.47, −1.79]	+0.52 [0.33, 0.71]	−17.55 [−20.14, −14.99]
Gated, seed 76022	−2.25 [−2.87, −1.62]	−0.18 [−0.40, 0.04]	−7.93 [−9.56, −6.28]
Gated, seed 76023	−2.24 [−2.91, −1.57]	−0.15 [−0.35, 0.06]	−7.60 [−8.95, −6.22]
Gated, seed 76024	−2.60 [−3.18, −2.00]	−0.22 [−0.42, −0.02]	−8.99 [−10.37, −7.65]

Table 32 separates two forms of failed consequence propagation on unseen edits. In the *too weak* stratum, reversing the edited-token state lowers the new-output margin for the execution-tuned base (-2.64) and all three attribution-gated seeds (-2.60 to -2.24): the edit contributes evidence in the correct direction even though it is insufficient to change the answer. In the *wrong direction* stratum, reverse replacement raises that margin for the execution-tuned base (+0.52), exposing an edited-token contribution in the wrong direction; the mean effect becomes negative in all three attribution-gated seeds (-0.22 to -0.15). The result refines the main diagnosis: an edit can register yet fail

either because its evidence is too weak or because downstream computation uses it in the wrong direction.

H RUNTIME

Table 33 reports total inference time for the same 32 short questions on one RTX 4090, after two warm-up passes and across five measured repetitions with rotated method order. Relative to the execution-tuned base, Counterfactual Regrounding changes latency by -0.5% at batch size 1 and $+0.4\%$ at batch size 8, while peak memory is 7.699 versus 7.701 GiB. It therefore preserves the single-forward-pass inference interface and adds essentially no overhead beyond the LoRA adapter already present in the execution-tuned base. During training only, retention adds one frozen-reference forward pass per context.

Table 33: Inference cost on an RTX 4090 (total seconds for 32 questions; peak memory).

Model	Batch 1 (s)	Batch 8 (s)	Memory (GiB)
Qwen3 backbone	1.252	0.420	7.690
Execution-tuned base	1.502	0.449	7.701
Revision-only	1.489	0.445	7.700
Ours	1.495	0.451	7.699

I EARLIER CHANGED-PAIR STUDY

Before introducing the joint Changed/Unchanged benchmark unit, we fixed two Changed-pair methods (Appendix B.4) and evaluated them on the 576 diagnostic pairs at three Qwen3 sizes (Table 34). Averaged across sizes, Revision rises from 10.9% for the execution-tuned bases to 16.3% with weighted supervision and 16.1% with pair loss. Weighted supervision also lowers inertia at all three sizes. Because this study contains no Unchanged input, it cannot establish selective semantic updating; its role is historical evidence that the simpler weighted objective matched the explicit pair loss on revision, motivating the weighting component later combined with Unchanged supervision and retention in Counterfactual Regrounding.

Table 34: Earlier Changed-pair methods on the 576 diagnostic pairs (%). Revision requires correct predictions for both program versions on the Changed input. Base is the execution-tuned base, and Weighted is weighted supervision. Inertia uses each method's own correct-original cohort.

	Revision			Inertia ↓	
Model	Base	Weighted	Pair loss	Base	Weighted
Qwen3-0.6B	5.0	8.0	7.6	85.1	79.6
Qwen3-1.7B	7.8	12.4	12.3	82.3	74.1
Qwen3-4B	19.8	28.3	28.5	65.7	53.8