How Much Evidence Should a Self-Correction Carry? Adaptive Dirichlet Evidence for Self-Distillation

Anonymous authors
Paper under double-blind review

Abstract

Language-model agents can generate, execute, and revise their own solutions, creating supervision for self-improvement. A correction's apparent quality, however, need not reveal how concentrated its external support is. We introduce *Effective-Evidence Self-Distillation* (EESD), which separates relative transition support from an effective pseudo-count mass derived from normalized execution relevance. A Dirichlet posterior converts these quantities into an uncertainty-penalized correction weight for KL-anchored self-distillation. Under a symmetric prior, changing mass preserves category ordering, while effective mass cannot increase the supervised coefficient relative to matched fixed-mass weighting. Across four model–domain history sweeps, increasing visible observations from one to eight reduces EED future-outcome NLL by 55.0–59.3%. At eight observations, EED also achieves lower NLL than fixed mass in all four comparisons. In a matched DeepSeek/RunBugRun study, the predictors agree in argmax on all 3,000 examples, with the largest NLL gain in the most concentrated relevance bin. The complete learning update achieves its largest observed downstream gain on DeepSeek/CodeARC, raising all-tests Pass@1 from 15.0% to 20.4%. These results connect an explicit evidence decomposition to probability estimation and correction learning.

1 Introduction

Language-model agents increasingly operate in environments that return objective feedback: compilers, unit tests, simulators, APIs, and interactive tools. This has made self-correction practical. Self-Refine and Reflexion reuse model-generated feedback or reflection at inference time (Madaan et al., 2023; Shinn et al., 2023); Self-Debugging revises programs after observing execution behavior (Chen et al., 2024); and execution-guided repair, search, and verification systems use tests to select or refine candidate programs (Ni et al., 2023; Tang et al., 2024; Li et al., 2025b;a; Tao et al., 2026). These methods answer an increasingly well-studied question: *can the model produce a better next answer after feedback?*

A second line of work asks which self-generated trajectories are safe or useful to learn from. Correctness filtering retains successful trajectories, reward-based methods favor higher-reward behavior, and confidence-aware self-training distinguishes trajectories that share an observed outcome (Jiang et al., 2025; Jang et al., 2025; Yang et al., 2025). These signals assess trajectory quality, but need not separately represent two properties of the execution record: *which before/after transitions the observations support* and *how much total evidence supports that direction*. The question is how the weight is constructed, not whether its final representation is scalar.

Figure 1 illustrates the distinction through an off-by-one repair. Extending a Python range endpoint makes all four public executions pass, so relative support is $r = (1, 0, 0, 0)$ under either relevance pattern. Uniform relevance assigns comparable weight to every execution and gives $E(p) = 4$; concentrated relevance $p = (0.97, 0.01, 0.01, 0.01)$ gives $E(p) \approx 1.062$. A fixed-mass control restores total mass to four in both cases. EESD instead lets relevance concentration change the supervised weight while retaining the same correction target. The relevance vectors are stipulated to isolate this distinction; the code and before/after outputs make its meaning concrete.

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.

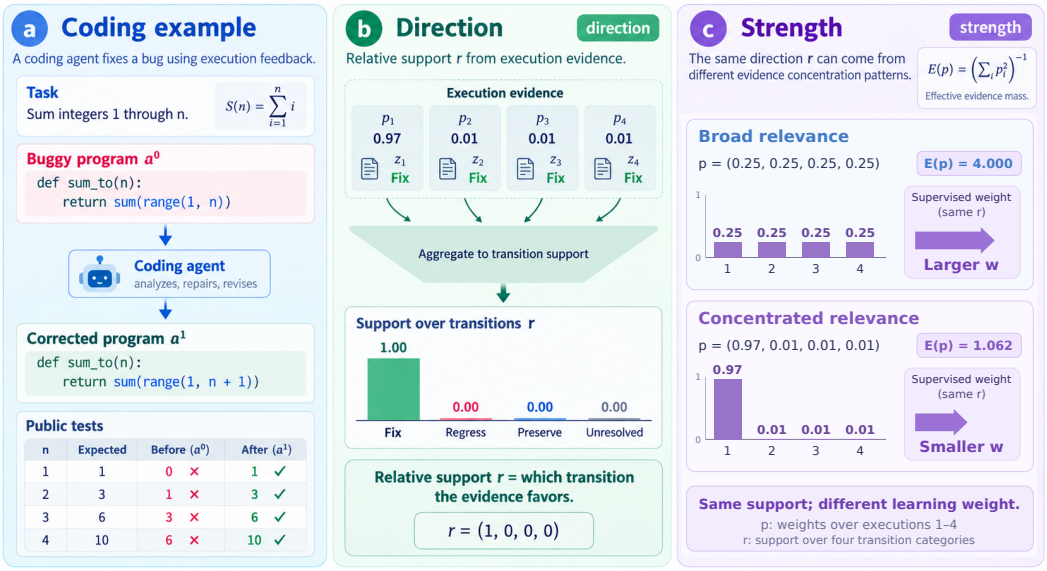


Figure 1: **Direction and strength in a coding correction.** (a) Fixing an off-by-one range error changes four public executions from fail to pass. (b) Aggregating execution relevance by transition gives $r = (1, 0, 0, 0)$: all support favors FIX. (c) The same transitions under uniform and concentrated relevance give $E(p) = 4.000$ and 1.062; at fixed prior and utility settings, they yield different supervised weights. The p bars index executions, whereas the r bars index transition categories. Relevance weights are stipulated for this analytical illustration, not measured on a benchmark.

The problem becomes more consequential when corrections are written back into model parameters. A mistaken inference-time correction affects one answer; an over-weighted training correction changes the policy that will generate future answers and future training data. We therefore ask: *given a self-generated correction, how much should the next policy learn from the execution evidence supporting it?*

We introduce *Effective-Evidence Self-Distillation* (EESD), a probabilistic trust layer between correction and learning. Given nonnegative relevance scores s_i, EESD normalizes them into p. Relative support r describes *what* transition categories the observations favor, while $E(p) = (\sum_i p_i^2)^{-1}$ describes *how much* pseudo-count mass that support contributes. A Dirichlet posterior produces an uncertainty-penalized correction utility; its positive part scales correction cross-entropy while a KL term anchors the student to the entering policy. The upstream correction generator and relevance estimator remain modular: the contribution is how their execution record is converted into a learning coefficient.

Our contributions are:

1. We formulate correction-level *direction–strength separation*: distinguishing relative execution support from the total evidence mass assigned to that support.

2. We introduce EED, which converts relevance concentration into adaptive Dirichlet pseudo-count mass, and connect its posterior utility to KL-anchored LoRA self-distillation. We characterize count recovery, ordering invariance, and the resulting control of the supervised coefficient.

3. We evaluate the construction through visible-history scaling, a matched-mass mechanism study with fixed argmax, and one-round fresh program generation across multiple models and domains.

Method	Inference revision / selection	Generator update	Execution feedback	Adaptive training	Transition support r	Effective mass $E(p)$	Support / mass separation
Self-Refine	✓	–	–	–	–	–	–
Reflexion (coding)	✓	–	✓	–	–	–	–
Self-Debugging	✓	–	✓	–	–	–	–
Execution-guided verification / reranking	✓	–	✓	–	–	–	–
Importance-weighted self-training	–	✓	–	✓	–	–	–
Confidence-aware self-training	–	✓	–	✓	–	–	–
Reward / process-weighted learning	○	○	✓	○	–	–	–
EESD (ours)	–	✓	✓	✓	✓	✓	✓

Table 1: **Positioning by the role of feedback.** ✓: explicit use; –: absent in the cited formulation; ○: varies within a grouped family. Generator updates exclude auxiliary-verifier training. Adaptive training includes filtering, preferences, and reward weighting. The last three columns concern our correction-level support–mass construction, not uncertainty modeling in general. This is a conceptual comparison, not a benchmark result; representative methods are discussed in Section 2.

2 Related Work

Table 1 positions EESD relative to representative self-correction and self-training approaches. The central distinction is between improving an answer, assessing its suitability for learning, and determining how much execution evidence supports the resulting update.

Self-correction and execution-guided repair. A broad line of work studies how language models can improve an answer after observing feedback. Self-Refine iteratively revises outputs using model-generated critiques, while Reflexion stores verbal feedback from previous attempts to guide subsequent decisions (Madaan et al., 2023; Shinn et al., 2023). Reflexion can use external execution feedback in its coding setting; Self-Debugging likewise uses execution behavior to revise programs (Chen et al., 2024). Execution-guided verification and reranking use execution results to assess and select candidate programs (Ni et al., 2023; Li et al., 2025a); related repair-search and critic-based systems use feedback to guide further revisions (Tang et al., 2024; Tao et al., 2026). LEVER trains an auxiliary verifier rather than the code-generating policy. REx already uses Bayesian beliefs, but to allocate repair-search effort rather than correction-level training weights. These approaches illustrate how external execution evidence can support inference-time correction and candidate selection. Our question begins after a correction has been obtained: rather than introducing another repair or reranking procedure, EESD determines how strongly that correction should contribute to a parameter update.

Learning from self-generated corrections. Self-training turns generated trajectories into parameter updates, making the reliability of each trajectory consequential beyond a single inference episode. Correctness filtering retains successful samples, while importance-weighted self-training adjusts the selection or contribution of generated examples under distributional mismatch (Jiang et al., 2025). Confidence-aware self-training uses model or reasoning confidence to distinguish trajectories that may share the same observed outcome (Jang et al., 2025). Reward / process-weighted learning uses outcome rewards or intermediate-step assessments: CodePRM learns a process evaluator, whereas ExecCritic also trains repair agents (Li et al., 2025b; Tao et al., 2026). Iterative preference-learning methods study how to sustain improvement across updates (Yang et al., 2025). These families, grouped in Table 1, refine which trajectories are selected or how they influence learning. EESD focuses on a different quantity within each correction record: the amount of execution evidence supporting its relative transition preferences. Its final training weight is also scalar, but is derived

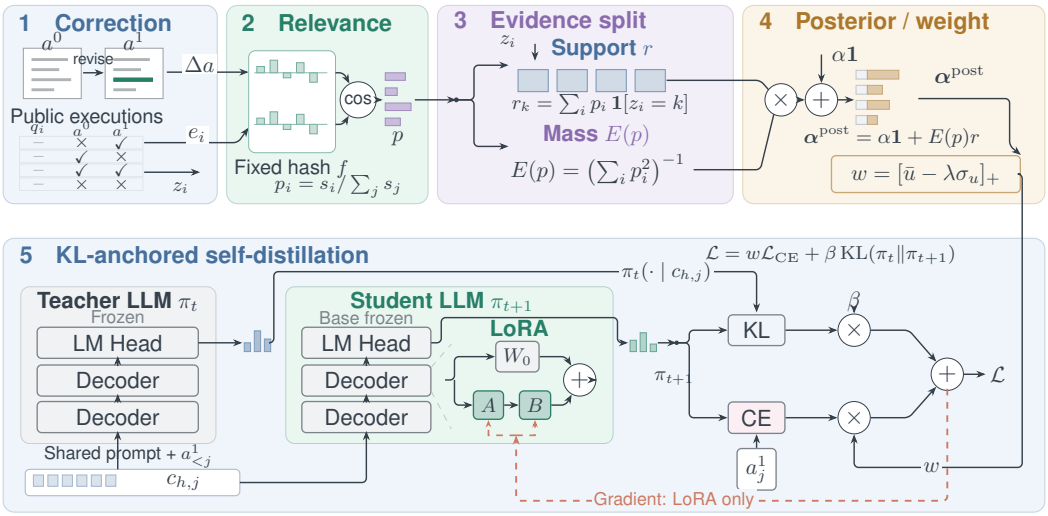


Figure 2: **Effective-Evidence Self-Distillation.** The public correction record yields p and transition labels z_i. Parallel support and mass summaries form a pseudo-count posterior and weight w. Teacher and student consume the same prompt and past correction tokens; the current target a_j^1 enters CE only. Teacher probabilities enter KL, not the student input. Only CE is weighted by w, and only LoRA receives gradients. Decoder stacks and the low-rank inset are schematic, not a target-layer specification.

from separately represented evidence direction and evidence amount, rather than identified with correctness, reward, or model confidence alone.

Evidential uncertainty and effective sample size. Our probabilistic construction connects to two established statistical ideas. Dirichlet models distinguish categorical probabilities from distributional concentration and have been used to represent evidential uncertainty (Sensoy et al., 2018). The inverse squared concentration of normalized weights is also a standard effective-sample-size functional (Martino et al., 2017). We introduce neither component. Instead, EESD uses relative transition support r to represent evidence direction and effective mass $E(p)$ to represent the amount of relevance-weighted support. Their direction–strength separation determines the Dirichlet pseudo-count update and its uncertainty-penalized correction utility. This utility sets the supervised coefficient in KL-anchored self-distillation: for EESD, adaptive training in Table 1 refers to this correction-specific coefficient, not the norm of an optimizer step. The distinction is therefore the explicit execution-to-learning decomposition, rather than the use of uncertainty modeling or adaptive weighting in general.

3 Effective-Evidence Self-Distillation

EESD converts execution feedback into a correction-specific training weight. The key is to separate *relative transition support* from *total evidence mass*, then use both to determine how strongly to learn from a correction.

3.1 Correction evidence

For a programming task x, policy π_t produces an initial program a^0, and an upstream repair procedure returns a correction a^1. Both programs are evaluated on the same $n \geq 1$ public executions $\{q_i\}_{i=1}^n$, yielding before/after outcomes (o_i^0, o_i^1). Let h denote this correction record. Each outcome pair defines a transition

$$z_i \in \{\textsc{Fix}, \textsc{Regress}, \textsc{Preserve}, \textsc{Unresolved}\}, \tag{1}$$

corresponding to fail→pass, pass→fail, pass→pass, and fail→fail, respectively.

Each execution receives a relevance score $s_i \geq 0$, with $\sum_i s_i > 0$. Define

$$p_i = \frac{s_i}{\sum_j s_j}, \tag{2}$$

$$r_k = \sum_i p_i \mathbb{I}[z_i = k]. \tag{3}$$

The four-vector r describes which transitions the observations support. Here direction denotes category ordering, not a parameter-space gradient. Our implementation obtains s_i from a frozen lexical kernel comparing the code change with public execution descriptors; Appendix B gives the details.

3.2 From effective evidence to correction utility

Our fixed-mass control uses $c_k^{\text{fixed}} = nr_k$, preserving relative support while assigning every record of length n the same total mass. EED replaces this nominal mass with

$$E(p) = \left(\sum_i p_i^2 \right)^{-1}, \tag{4}$$

$$\alpha_k^{\text{post}} = \alpha + E(p) r_k, \qquad \alpha > 0, \tag{5}$$

and defines $\theta \mid h \sim \text{Dir}(\boldsymbol{\alpha}^{\text{post}})$. Thus r allocates pseudo-counts across transitions, while $E(p)$ sets their total mass. We have $1 \leq E(p) \leq n$; uniform relevance gives $E(p)p_i = 1$ and recovers ordinary counts. The Dirichlet is a working uncertainty model conditional on relevance: its pseudo-count mass measures concentration, not the number of statistically independent tests.

To reward fixes and penalize regressions, set $v = (1, -1, 0, 0)$. Writing $\alpha_0 = \sum_k \alpha_k^{\text{post}}$ and $\mu_k = \alpha_k^{\text{post}} / \alpha_0$, the posterior utility moments are

$$\bar{u} = \mathbb{E}[v^\top \theta] = \mu_{\text{FIX}} - \mu_{\text{REGRESS}}, \tag{6}$$

$$\sigma_u^2 = \frac{\mu_{\text{FIX}} + \mu_{\text{REGRESS}} - \bar{u}^2}{\alpha_0 + 1}. \tag{7}$$

The correction weight is

$$w(h) = [\bar{u} - \lambda \sigma_u]_+, \qquad \lambda \geq 0, \tag{8}$$

where $[a]_+ = \max(a, 0)$. The uncertainty penalty reduces the supervised contribution of corrections with less decisive posterior utility; nonpositive utility yields zero supervised weight. It is not a calibrated confidence bound.

For the all-FIX example in Figure 1, $\alpha = 0.1$ and $\lambda = 0.5$ give weights 0.832 and 0.542 for uniform and concentrated relevance. The matched fixed-mass control assigns 0.832 to both. Appendix G.1 gives the calculation. These are supervised coefficients, not measured performance or optimizer-step norms.

3.3 KL-anchored self-distillation

The student learns from a^1 while a frozen copy of π_t supplies a forward-KL anchor:

$$\mathcal{L}_{\text{EESD}}(h) = w(h)\mathcal{L}_{\text{CE}}(h) + \beta D_{\text{KL}}^{(h)}(\pi_t \| \pi_{t+1}). \tag{9}$$

Here $\mathcal{L}_{\text{CE}}$ is correction cross-entropy, and $D_{\text{KL}}^{(h)}$ is forward KL averaged over the same response-token prefixes. Only the supervised term is weighted by $w(h)$. We train LoRA parameters with the quantized base frozen; token-level definitions and fixed hyperparameters are in Appendix B.

Proposition 1 (Outcome ordering and correction weight)**.** *Fix r, a symmetric prior $\alpha > 0$, $\lambda \geq 0$, and $v = (1, -1, 0, 0)$. Replace $E(p)$ by any mass $\tau > 0$ in Equations 5–8. The posterior mean satisfies*

$$\mu_k(\tau) = \gamma(\tau) r_k + \frac{1 - \gamma(\tau)}{4}, \qquad \gamma(\tau) = \frac{\tau}{4\alpha + \tau}. \tag{10}$$

Consequently, category ordering and the argmax set are invariant to τ. Moreover, the correction weight w_τ is nondecreasing in τ, so

$$w_{E(p)} \leq w_n. \tag{11}$$

Thus effective evidence cannot increase the supervised coefficient relative to matched fixed-mass weighting. Probability estimates can nevertheless change without changing categorical predictions; whether this improves NLL depends on the observed outcome. Appendix A gives the proofs and the log-loss characterization.

4 EXPERIMENTS

We evaluate three questions. **RQ1** examines whether additional visible executions improve outcome prediction. **RQ2** isolates how effective mass changes predictive probabilities at fixed relative support. **RQ3** tests the complete correction-learning update through fresh program generation. Figures 3 and 4 organize these measurements, with full numerical reports in the appendix.

4.1 EXPERIMENTAL SETUP

Tasks and models. We use RunBugRun and CodeARC, together with APPS Replay and CodeContests Replay (Prenner & Robbes, 2023; Wei et al., 2025). RunBugRun contains executable buggy programs; CodeARC tests program synthesis from behavioral observations. All results use the recorded source populations and execution budgets below; the replay domains use controlled synthesis extensions rather than official leaderboard settings. The CodeARC evaluation also differs from its original interactive differential-testing protocol (Wei et al., 2025). The downstream matrix covers Qwen2.5-Coder-7B-Instruct, DeepSeek-Coder-6.7B-Instruct, and Gemma-3-4B-it, with 500 primary sources per model–domain cell.

Measurements and comparisons. The prediction studies measure future-outcome negative log-likelihood (NLL) and categorical accuracy. The history sweep compares fixed mass n with effective mass $E(p)$ as visible history grows. A separate matched comparison holds relative support and predictor parameters fixed, changing only the pseudo-count mass. Downstream evaluation generates one fresh candidate per source after one update; all-tests Pass@1 requires passing all ten evaluator tests. It compares full EESD (`eesd_full`) with the unchanged entering policy (`no_update`). The paired effect, in percentage points, is

$$\Delta_{\text{pp}} = 100\big[S(\texttt{eesd_full}) - S(\texttt{no_update})\big], \tag{12}$$

where S is the fraction of solved sources. We retain the reported paired 95% intervals from 10,000 whole-source bootstrap draws. These quantify source-level uncertainty for the evaluated policies. Hidden outcomes are evaluation signals, not inputs to the evidence scorer. Fresh generation means a new completion on the recorded source population; protocol and replication details are in Appendix B.

Training and hardware. We adapt LoRA parameters on a frozen 4-bit NF4 base with bfloat16 computation. The correction scorer uses $\alpha = 0.1$ and $\lambda = 0.5$; the forward-KL coefficient is $\beta = 0.03$. Experiments use NVIDIA RTX A6000 hardware with 48 GB nominal memory per GPU. Evidence weights are computed from public records before student fitting, and the frozen entering policy supplies the KL reference. Appendix B specifies the relevance kernel, response-token objective, and optimization settings.

4.2 RQ1: DOES ADDITIONAL VISIBLE EVIDENCE IMPROVE PREDICTION?

We first ask whether the available executions carry useful predictive information. Figure 3 varies visible history size over $n \in \{1, 2, 4, 8\}$ for DeepSeek and Qwen on RunBugRun and CodeARC. Here n counts observations, not training rounds. This study establishes the value of accumulating execution history before examining how its mass should be assigned.

Prediction improves with longer histories. EED NLL decreases at every measured step in all four model–domain cells. On RunBugRun, the endpoint changes are $0.4696 \to 0.2114$ for DeepSeek and $0.4737 \to 0.2113$ for Qwen. On CodeARC, they are $0.7103 \to 0.2892$ and $0.7785 \to 0.3415$. Table 2 expresses these changes as relative NLL reductions of 55.0–59.3%. Categorical accuracy improves as well: by 2.5 and 1.9 percentage points on RunBugRun, and by 5.5 and 6.1 points on CodeARC, respectively. Additional history therefore benefits both the probability assigned to

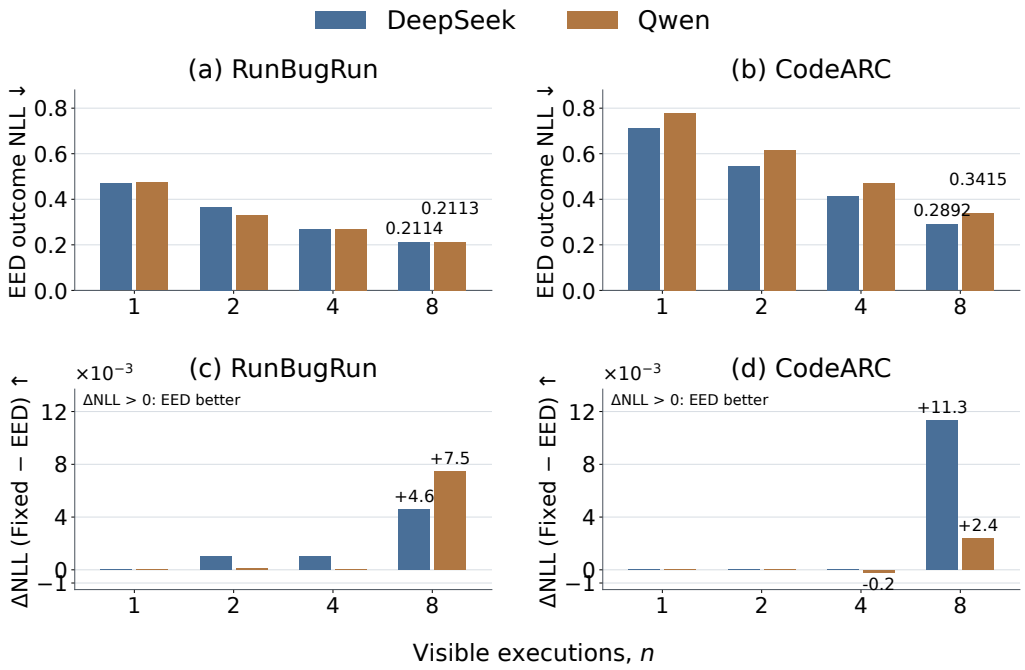


Figure 3: **History size and effective evidence answer different questions.** (a,b) EED future-outcome NLL: lower is better. (c,d) $\Delta\text{NLL} = \text{NLL}_{\text{Fixed}} - \text{NLL}_{\text{EED}}$: higher is better, in units of 10^{-3}. All panels use $n = 1, 2, 4, 8$ visible observations, not training rounds. Differences are calculated from the four-decimal measurements in Table 8. The matched-parameter comparison is reported separately in Section 4.3.

Table 2: **Endpoint summary of the EED history sweep.** NLL reduction and accuracy change compare $n = 1$ with $n = 8$; $\Delta\text{NLL}_8 = \text{NLL}_{\text{Fixed}} - \text{NLL}_{\text{EED}}$ at $n = 8$. The reductions are calculated from the reported rounded measurements.

Benchmark	Backbone	EED NLL $1 \to 8$	Reduction (%)	Acc. change (pp)	$\Delta\text{NLL}_8 \uparrow$
RunBugRun	DeepSeek 6.7B	$0.4696 \to 0.2114$	55.0	+2.5	+0.0046
RunBugRun	Qwen2.5 7B	$0.4737 \to 0.2113$	55.4	+1.9	+0.0075
CodeARC	DeepSeek 6.7B	$0.7103 \to 0.2892$	59.3	+5.5	+0.0113
CodeARC	Qwen2.5 7B	$0.7785 \to 0.3415$	56.1	+6.1	+0.0024

the observed outcome and the categorical prediction. These measurements concern the outcome estimator; the learned policy is evaluated separately in RQ3.

Effective mass adds a distinct comparison. Fixed-mass prediction also benefits from additional observations. At $n = 8$, EED further reduces NLL relative to Fixed in all four cells, as summarized in Figure 4(a): 0.0046 and 0.0075 on RunBugRun, and 0.0113 and 0.0024 on CodeARC, for DeepSeek and Qwen respectively. These are absolute NLL differences, not accuracy percentage points. At $n = 1$, the two constructions coincide because $p_1 = 1$ and $E(p) = n = 1$; differences at shorter histories are generally small. Figure 3 and Appendix E retain the complete trajectories. Increasing n can change both relative support and effective mass, so this sweep should be distinguished from the fixed-support comparison that follows.

4.3 RQ2: Does effective evidence change strength without changing direction?

The primary matched comparison uses DeepSeek/RunBugRun with the same relative support and effective-arm hyperparameters for both predictors. Only total pseudo-count mass changes. Their categorical argmax predictions agree on all 3,000 future-outcome examples, while effective mass

Table 3: **Concentration-stratified matched-mass comparison.** Primary DeepSeek/RunBugRun comparison. Gain is fixed-mass NLL minus EED NLL. Both predictors agree in argmax on all 3,000 examples.

Effective-mass bin	Examples	Mean $E(p)$	NLL gain ↑
$[1, 2)$	568	1.511	**+0.0159**
$[2, 3)$	815	2.521	+0.0077
$[3, 4]$	1617	3.681	+0.0018

reduces aggregate NLL by 0.00609 with a 95% source-bootstrap interval of $[0.00211, 0.01075]$. Thus better probability estimates do not require changing the selected category. The ordering argument in Proposition 1 applies to any categorical predictor with a symmetric prior, as detailed in Appendix A.4; the four-transition utility is the separate training construction.

The largest gain occurs under concentrated relevance. Table 3 and Figure 4(b) divide the matched comparison into three effective-mass bins. The NLL reduction is 0.0159 for $1 \leq E(p) < 2$, 0.0077 for $2 \leq E(p) < 3$, and 0.0018 for $3 \leq E(p) \leq 4$. The corresponding mean masses retain 37.8%, 63.0%, and 92.0% of the nominal count of four (Appendix G). In this comparison, the largest predictive benefit is associated with the largest reduction in pseudo-count mass. When support is already broadly distributed, effective and nominal mass are closer and the observed difference is smaller.

The intervention neither finds another candidate nor supplies additional executions. It changes shrinkage toward the symmetric prior while retaining the relative support and its category ordering. This is the probability-level signature targeted by the method: the favored outcome can remain fixed while the confidence assigned to it changes. The mass fractions describe pseudo-counts, not measured optimizer-step sizes.

From probability estimates to learning. The matched study tests the evidence estimator rather than policy adaptation. For training, the posterior is further mapped to a correction-specific utility and supervised coefficient. Proposition 1 characterizes this mapping; RQ3 examines the complete resulting update. The other recorded mechanism repetitions and their intervals are provided in Appendix F, separately from the history sweep. The log-loss analysis in Appendix A.4 explains why a change in mass need not have the same benefit for every outcome.

4.4 RQ3: Does correction learning improve fresh generation?

We now move from outcome prediction to the code-generating policy. Each evaluated program is freshly sampled after learning, rather than chosen by reranking an existing candidate bank. Figure 4(c) presents four selected model–domain contrasts against the unchanged policy; Appendix D reports the complete twelve-cell evaluation. The displayed contrasts are the cells with positive point estimates, not an aggregate over the matrix.

The clearest gain is on DeepSeek/CodeARC. All-tests Pass@1 rises from $75/500$ (15.0%) to $102/500$ (20.4%), a net increase of 27 solved sources. The paired improvement is $+5.4$ percentage points, with a 95% source-bootstrap interval of $[+2.8, +8.0]$. This result concerns complete programs passing all ten evaluator tests, not individual test outcomes or the accuracy of an auxiliary predictor. It demonstrates a benefit of the complete correction-learning update in this model–domain setting.

Further gains appear on RunBugRun and CodeContests Replay. DeepSeek increases from $184/500$ to $198/500$ on RunBugRun ($+2.8$ points), and Qwen increases from $228/500$ to $240/500$ ($+2.4$ points). DeepSeek also increases from $18/500$ to $24/500$ on CodeContests Replay ($+1.2$ points). These correspond to 14, 12, and 6 additional solved sources in the respective aggregate counts. Their paired intervals include zero, so they provide positive point estimates rather than independent claims of statistically resolved improvement. The full matrix shows that the effect remains model–domain dependent.

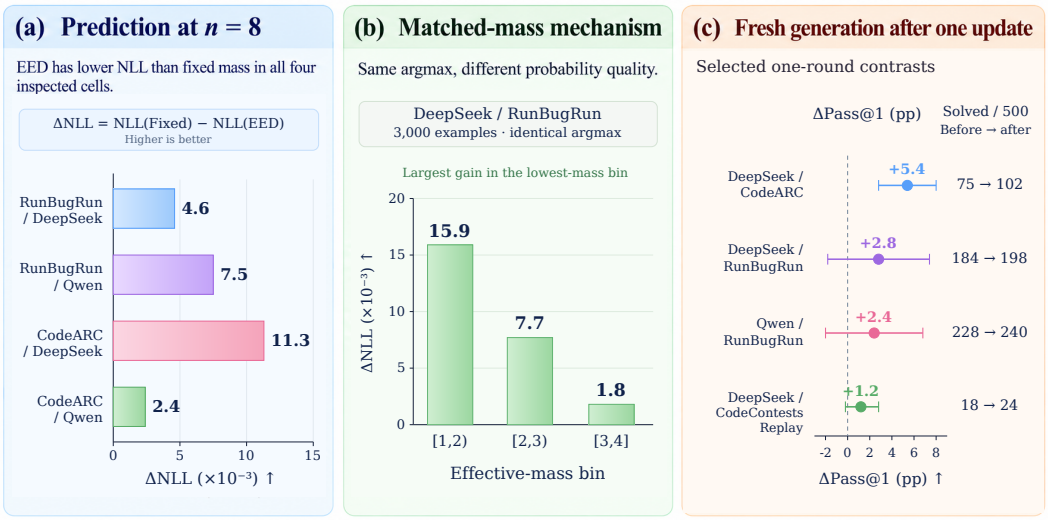


Figure 4: **Prediction and fresh-generation performance.** (a) Fixed-minus-EED NLL at $n = 8$ in the four history-sweep cells. (b) The matched DeepSeek/RunBugRun comparison by effective-mass bin; argmax agrees on all 3,000 examples. Both NLL panels use units of 10^{-3} and positive values favor EED. (c) Four selected one-round contrasts against `no_update`, with paired 95% source-bootstrap intervals and solved counts out of 500. The complete model–domain evaluation is in Table 7.

Together, the studies examine successive parts of the evidence-to-learning path: additional observations improve prediction, matched mass changes probability estimates without changing category ordering, and the complete weighted update improves fresh generation in several model–domain settings. The downstream comparison assesses EESD as a whole; the controlled probability-level result is not a component-level training ablation. Appendix H specifies the comparisons represented in the current result matrix.

5 Limitations

The evaluation focuses on executable coding tasks and one-round adaptation. More expressive relevance estimators can replace the frozen lexical kernel without changing the construction, but effective mass measures concentration rather than test independence. The reported downstream comparison evaluates the complete update; component attribution and source-disjoint transfer require separate evidence. Broader tasks and repeated adaptation remain natural extensions. Full results and replication details are provided in the appendix.

6 Conclusion

EESD separates what execution evidence supports from how much pseudo-count mass it contributes, then uses posterior utility to weight KL-anchored self-distillation. The analysis connects this decomposition to category-order invariance and the supervised coefficient. Experiments show substantial predictive gains as visible history grows, lower NLL than fixed mass in all four eight-observation comparisons, and a matched-mass improvement without an argmax change. The complete update raises DeepSeek/CodeARC all-tests Pass@1 by 5.4 percentage points, alongside additional positive point estimates on RunBugRun and CodeContests Replay. These findings support treating a correction's learning weight as a function of its supporting execution evidence, rather than visible success alone.

AI USE STATEMENT

Generative AI tools were used for method discussion, mathematical formulation, software development and debugging, literature discovery and summarization, drafting and revising manuscript text, reference formatting, and preparation of figure specifications. All numerical claims in the manuscript are restricted to experiment artifacts or calculations derived from those artifacts; AI-generated text is not treated as empirical evidence. The authors remain responsible for verifying the mathematics, citations, code, experimental provenance, and final manuscript before submission.

REPRODUCIBILITY STATEMENT

Appendix B.5 states the correction-learning algorithm, and Appendix B specifies the relevance kernel, token-level objective, and recorded training settings. Appendix C defines the candidate-bank diagnostic, and Appendix D reports the complete downstream matrix; Appendices E and F provide history-size results and matched-mass sensitivity across recorded repetitions. Appendix A gives the derivations. The empirical scope and comparisons not included in the reported matrix are stated in Appendix H.

ETHICS STATEMENT

Generated code must be executed in controlled environments with resource, timeout, and filesystem restrictions. Passing a benchmark test suite is not a security or correctness guarantee. Self-training can amplify unsafe or insecure behavior when feedback is incomplete; evidence-aware weighting does not provide a security guarantee.

REFERENCES

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2304.05128.

Hyosoon Jang, Yunhui Jang, Sungjae Lee, Jungseul Ok, and Sungsoo Ahn. Self-training large language models with confident reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 14925–14939, 2025. doi: 10.18653/v1/2025.findings-emnlp.806.

Chunyang Jiang, Chi-Min Chan, Wei Xue, Qifeng Liu, and Yike Guo. Importance weighting can help large language models self-improve. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(23):24257–24265, 2025. doi: 10.1609/aaai.v39i23.34602.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, and Ion Stoica. S*: Test time scaling for code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 15964–15978, 2025a. doi: 10.18653/v1/2025.findings-emnlp.865.

Qingyao Li, Xinyi Dai, Xiangyang Li, Weinan Zhang, Yasheng Wang, Ruiming Tang, and Yong Yu. Codeprm: Execution feedback-enhanced process reward model for code generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 8169–8182, 2025b. doi: 10.18653/v1/2025.findings-acl.428.

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

Luca Martino, Víctor Elvira, and Francisco Louzada. Effective sample size for importance sampling based on discrepancy measures. *Signal Processing*, 131:386–401, 2017. doi: 10.1016/j.sigpro.2016.08.025.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-Tau Yih, Sida Wang, and Xi Victoria Lin. Lever: Learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 26106–26128. PMLR, 2023.

Julian Aron Prenner and Romain Robbes. Runbugrun—an executable dataset for automated program repair, 2023.

Murat Sensoy, Lance Kaplan, and Melih Kandemir. Evidential deep learning to quantify classification uncertainty. In *Advances in Neural Information Processing Systems*, 2018.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Hao Tang, Keya Hu, Jin Peng Zhou, Sicheng Zhong, Wei-Long Zheng, Xujie Si, and Kevin Ellis. Code repair with llms gives an exploration-exploitation tradeoff. In *Advances in Neural Information Processing Systems*, 2024. doi: 10.52202/079017-3746.

Leitian Tao, Baolin Peng, Haorui Wang, Hang Wang, Hao Cheng, Wenlin Yao, Qianhui Wu, Tao Ge, Sharon Li, and Jianfeng Gao. Execcritic: Learn to test, test to improve for coding agents, 2026.

Anjiang Wei, Tarun Suresh, Jiannan Cao, Naveen Kannan, Yuheng Wu, Kai Yan, Thiago S. F. X. Teixeira, Ke Wang, and Alex Aiken. CodeARC: Benchmarking reasoning capabilities of LLM agents for inductive program synthesis, 2025. URL https://arxiv.org/abs/2503.23145.

Haoyan Yang, Ting Hua, Shangqian Gao, Binfeng Xu, Zheng Tang, Jie Xu, Hongxia Jin, and Vijay Srinivasan. Dynamic noise preference optimization for llm self-improvement via synthetic data, 2025.

A Proofs and Interpretation of Effective Evidence

A.1 Pseudo-count construction and elementary properties

Weighted-likelihood representation. Condition on the observed relevance weights p. Starting from a symmetric Dirichlet prior, the weighted likelihood

$$\widetilde{L}(\theta; h) = \prod_{i=1}^{n} \theta_{z_i}^{E(p)p_i} = \prod_{k=1}^{4} \theta_k^{E(p)r_k} \tag{13}$$

gives

$$\widetilde{p}(\theta \mid h) \propto \prod_{k=1}^{4} \theta_k^{\alpha-1+E(p)r_k}. \tag{14}$$

This is the density of $\mathrm{Dir}(\alpha\mathbf{1} + E(p)r)$. The construction is an algebraic weighted-likelihood interpretation of the pseudo-count update; it does not assert that the public executions are independent draws from a categorical sampling model.

Bounds and count recovery. Since $\sum_i p_i = 1$ and $p_i \geq 0$, Cauchy–Schwarz yields

$$1 = \left(\sum_i p_i\right)^2 \leq n \sum_i p_i^2,$$

while $\sum_i p_i^2 \leq 1$. Therefore $1 \leq E(p) \leq n$. Equality at the upper bound requires uniform weights, for which $E(p)p_i = 1$ and

$$E(p)r_k = \sum_i \mathbb{I}[z_i = k].$$

Equality at the lower bound requires a point mass. Multiplying every s_i by the same positive constant leaves p, r, $E(p)$, and the pseudo-count posterior unchanged.

A.2 Posterior utility moments

For a Dirichlet vector with mean μ and total concentration α_0,

$$\mathrm{Cov}(\theta) = \frac{\mathrm{diag}(\mu) - \mu\mu^\top}{\alpha_0 + 1}.$$

Thus $\mathbb{E}[v^\top\theta] = v^\top\mu$ and

$$\mathrm{Var}(v^\top\theta) = \frac{\sum_k \mu_k v_k^2 - (v^\top\mu)^2}{\alpha_0 + 1}.$$

Taking $v = (1, -1, 0, 0)$ gives Equations 6 and 7. The score $\bar{u} - \lambda\sigma_u$ is an uncertainty penalty, not a confidence bound with a claimed coverage level.

A.3 Proof of Proposition 1

Proof. Set $A = 4\alpha$. The posterior mean is

$$\mu_k(\tau) = \frac{\alpha + \tau r_k}{A + \tau} = \frac{\tau}{A + \tau} r_k + \frac{A}{A + \tau}\frac{1}{4}.$$

For any categories j, k,

$$\mu_j(\tau) - \mu_k(\tau) = \frac{\tau(r_j - r_k)}{A + \tau}.$$

Its sign is independent of $\tau > 0$, proving invariance of category ordering, including ties.

To establish monotonicity of the training weight, write

$$d = r_{\textsc{Fix}} - r_{\textsc{Regress}}, \qquad b = r_{\textsc{Fix}} + r_{\textsc{Regress}}.$$

Then

$$\bar{u}_\tau = \frac{\tau d}{A+\tau}, \tag{15}$$

$$\sigma_\tau^2 = \frac{(2\alpha+\tau b)(A+\tau)-\tau^2 d^2}{(A+\tau)^2(A+\tau+1)}. \tag{16}$$

If $d \le 0$, then $\bar{u}_\tau \le 0$ and $w_\tau = 0$ for every τ.

Suppose $d > 0$. All Dirichlet parameters are positive, so $\sigma_\tau > 0$. The ratio $T_\tau = \bar{u}_\tau/\sigma_\tau$ satisfies

$$T_\tau^2 = \frac{d^2(A+\tau+1)}{b-d^2+(2\alpha+Ab)/\tau+2\alpha A/\tau^2}. \tag{17}$$

Because $|d| \le b \le 1$, we have $b - d^2 \ge 0$. The numerator is strictly increasing in τ, and the positive denominator is strictly decreasing. Therefore T_τ is strictly increasing. Since $\bar{u}_\tau$ is also positive and increasing,

$$w_\tau = \bar{u}_\tau \left[1 - \frac{\lambda}{T_\tau}\right]_+$$

is nondecreasing for every $\lambda \ge 0$. Finally, $E(p) \le n$ gives $w_{E(p)} \le w_n$. □

This result concerns the coefficient of the supervised term under fixed r, α, λ, and utility vector. It does not bound the norm of an optimizer step or guarantee an improvement in the updated policy.

A.4 When reducing evidence mass changes log loss

For a predictive label set of size K, the symmetric-prior construction is $\mu_k(\tau) = (\alpha+\tau r_k)/(K\alpha+\tau)$. The difference $\mu_j - \mu_k = \tau(r_j - r_k)/(K\alpha+\tau)$ preserves category ordering for every K. This probability-level statement does not assume that a predictor's labels are the four training transitions. The correction-utility result above specifically uses $K = 4$ and $v = (1, -1, 0, 0)$.

For an observed transition y, define

$$\ell_\tau(y) = -\log \mu_y(\tau) = -\log \frac{\alpha+\tau r_y}{4\alpha+\tau}.$$

Differentiation gives

$$\frac{\partial \ell_\tau(y)}{\partial \tau} = \frac{\alpha(1-4r_y)}{(\alpha+\tau r_y)(4\alpha+\tau)}. \tag{18}$$

Consequently, when $E(p) < n$, reducing mass from n to $E(p)$ lowers log loss exactly when $r_y < 1/4$, increases it when $r_y > 1/4$, and leaves it unchanged when $r_y = 1/4$. Adaptive mass therefore has no universal NLL advantage, despite preserving the categorical argmax. Empirical gains depend on how relevance concentration aligns with errors in relative support.

A.5 Interpretive limits of effective mass

Consider replacing every observation by m identical copies. Each copy then has normalized weight p_i/m, so

$$E(p') = \left(\sum_i \sum_{\ell=1}^{m} (p_i/m)^2\right)^{-1} = mE(p), \qquad r' = r.$$

Thus effective mass does not remove duplication or statistical dependence. Likewise, scale invariance means it does not measure the absolute reliability of the raw relevance scores. Its role is specifically to turn normalized relevance concentration into an adaptive pseudo-count mass.

Table 4: **Distinct evaluation objects.** Candidate selection, outcome prediction, and post-update generation are not interchangeable measurements.

Study	Recorded protocol	Reported outcome
Candidate-bank audit	500 source groups per benchmark; eight candidates and four public executions per group	Visible-selected and retrospective hidden-oracle success rates
History-size prediction	$n \in \{1, 2, 4, 8\}$ visible observations; RunBugRun and CodeARC; DeepSeek and Qwen; primary run	Future-outcome NLL and categorical accuracy
Matched-mass prediction	DeepSeek/RunBugRun; same relative support and effective-arm settings within each contrast	NLL difference and argmax agreement; three repetitions
One-round learning	Three backbones and four domains; 500 primary sources per cell; one fitting run and one fresh candidate per source	Pass@1 requiring all ten evaluator tests; paired source-bootstrap interval

B Execution Protocol and Implementation

B.1 Measurement protocols

The retrospective hidden oracle in Appendix C uses evaluator outcomes only for selection within a fixed candidate bank. It is not a training control. The unchanged policy is the downstream reference; it is not a compute-matched correction-training baseline. NLL values concern categorical outcome prediction, whereas downstream Pass@1 concerns complete generated programs. The aggregate reports do not establish the relation between training-source and evaluator-source populations; we therefore make no claim of source-disjoint task transfer.

B.2 Randomness and uncertainty

The recorded random-number-generator seeds are 1701 for the primary history sweep, matched-mass comparison, and downstream evaluation, and 1702 and 1703 for the additional matched-mass repetitions. Each downstream model–domain cell reports one fitting run; its source-bootstrap interval is conditional on the evaluated policies and does not estimate variation across independent training runs. We retain the reported 10,000-draw paired intervals without recomputing them from marginal solved counts. The per-cell 95% intervals have not been adjusted for multiple comparisons.

B.3 Relevance scoring and token-level objective

Let Δa be the unified diff between a^0 and a^1, and let f be the frozen 256-dimensional signed-hash text encoder. Execution descriptor e_i contains the input, expected output, actual output, stderr, and categorical outcome. The one-round correction scorer uses

$$s_i = \exp\big(\kappa \cos\big(f(\Delta a), f(e_i)\big)\big), \qquad \kappa = 16. \tag{19}$$

No relevance model is trained, and hidden execution outcomes are excluded. The kernel configuration stated here is the correction scorer; it should not be substituted for unspecified settings in separate mechanism reports.

For scored correction-response length L_h, let $c_{h,j}$ be the actual training prompt followed by $a^1_{<j}$. The two token-mean losses are

$$\mathcal{L}_{\mathrm{CE}}(h) = -\frac{1}{L_h} \sum_{j=1}^{L_h} \log \pi_{t+1}(a^1_j \mid c_{h,j}), \tag{20}$$

$$D^{(h)}_{\mathrm{KL}}(\pi_t \| \pi_{t+1}) = \frac{1}{L_h} \sum_{j=1}^{L_h} D_{\mathrm{KL}}(\pi_t(\cdot \mid c_{h,j}) \,\|\, \pi_{t+1}(\cdot \mid c_{h,j})). \tag{21}$$

Teacher and student use the same response prefixes. The full execution record determines $w(h)$ but is not itself a token prefix. A valid zero-weight correction remains in the scored record: it removes the supervised pull, not the KL term.

B.4 Optimization settings

Table 5: **Recorded one-round training configuration.** GPU memory is nominal capacity, not measured peak usage.

Setting	Value
Hardware	NVIDIA RTX A6000; 48 GB per GPU
Student adaptation / base	LoRA only / frozen
Quantization	4-bit NF4, double quantization
Compute dtype	bfloat16
LoRA rank / scaling alpha / dropout	16 / 32 / 0
Optimizer / learning rate	AdamW / 2×10^{-4}
Gradient accumulation / clipping	16 / 1.0
Maximum sequence length	4608
Relevance encoder / κ	256-dimensional signed hash / 16
Dirichlet prior α	0.1
Utility / uncertainty coefficient	$(1, -1, 0, 0)$ / $\lambda = 0.5$
Forward-KL coefficient β	0.03
Loss normalization	Response tokens
Training budget policy	Response-token accounting

B.5 One-correction algorithm

Given (a^0, a^1) and their public before/after execution outcomes:

1. Compute s_i using Equation 19 and normalize $p_i = s_i / \sum_j s_j$.
2. Assign each outcome pair its transition z_i and aggregate $r_k = \sum_i p_i \mathbb{I}[z_i = k]$.
3. Set $E(p) = 1/\sum_i p_i^2$ and $\alpha_k^{\text{post}} = \alpha + E(p) r_k$.
4. Calculate $\bar{u}$ and σ_u under $v = (1, -1, 0, 0)$, then set $w = [\bar{u} - \lambda \sigma_u]_+$.
5. Fit the LoRA student with $w\mathcal{L}_{\text{CE}} + \beta D_{\text{KL}}^{(h)}(\pi_t \| \pi_{t+1})$ while the entering-policy teacher is frozen.

C Candidate-Bank Selection Diagnostic

This diagnostic is separate from the learning method. For each of 500 source groups per benchmark, visible-score selection chooses among the same eight candidates using four public executions. A *retrospective hidden oracle* instead selects the candidate with the highest additional evaluator score from that fixed bank. It is not available to the learner and provides only an empirical upper bound on selection within the already generated candidates.

Table 6: **Selection headroom within a fixed candidate bank.** The retrospective oracle is diagnostic only and does not generate candidates, score evidence, or train the policy.

Benchmark	Visible selected (%)	Hidden oracle (%)	Gap (pp)
CodeARC	27.0	31.2	+4.2
RunBugRun	71.4	75.4	+4.0

The gaps quantify selection headroom within the fixed bank; they do not isolate evidence mass as its cause.

Table 7: Complete one-round downstream matrix for the primary run. Each cell contains 500 primary sources and reports fresh all-tests Pass@1.

Benchmark	Backbone	`no_update` ↑	EESD ↑	Δ (pp) ↑	95% CI (pp)
RunBugRun	Qwen2.5 Coder 7B	228/500	240/500	+2.4	[-2.0,+6.8]
	DeepSeek Coder 6.7B	184/500	198/500	+2.8	[-1.8,+7.4]
	Gemma 3 4B	142/500	132/500	-2.0	[-5.6,+1.4]
CodeARC	Qwen2.5 Coder 7B	85/500	83/500	-0.4	[-2.8,+2.0]
	DeepSeek Coder 6.7B	75/500	**102/500**	**+5.4**	**[+2.8,+8.0]**
	Gemma 3 4B	59/500	56/500	-0.6	[-2.6,+1.4]
APPS Replay	Qwen2.5 Coder 7B	57/500	57/500	+0.0	[-1.8,+1.8]
	DeepSeek Coder 6.7B	23/500	17/500	-1.2	[-3.2,+0.8]
	Gemma 3 4B	36/500	2/500	-6.8	[-9.0,-4.8]
CodeContests Replay	Qwen2.5 Coder 7B	47/500	39/500	-1.6	[-3.2,+0.0]
	DeepSeek Coder 6.7B	18/500	24/500	+1.2	[-0.2,+2.8]
	Gemma 3 4B	31/500	4/500	-5.4	[-7.6,-3.2]

Table 8: Primary history-size sweep with both fixed and effective mass. Values are future-outcome NLL; lower is better. Displayed precision is four decimals.

		$n = 1$		$n = 2$		$n = 4$		$n = 8$	
Benchmark	Backbone	Fixed	EED	Fixed	EED	Fixed	EED	Fixed	EED
RunBugRun	DeepSeek	.4696	.4696	.3656	.3646	.2703	.2693	.2160	**.2114**
RunBugRun	Qwen	.4737	.4737	.3292	.3291	.2704	.2704	.2188	**.2113**
CodeARC	DeepSeek	.7103	.7103	.5440	.5440	.4149	.4149	.3005	**.2892**
CodeARC	Qwen	.7785	.7785	.6176	.6176	.4725	.4727	.3439	**.3415**

D Complete One-Round Downstream Results

The complete matrix is heterogeneous: DeepSeek has positive point estimates in three of four domains, Qwen improves on RunBugRun and is unchanged on APPS Replay, and Gemma decreases in all four cells. The intervals are paired source-bootstrap intervals for one fitting run per cell, not intervals across independent training runs. We therefore do not claim uniform dominance.

E History-Size Mechanism Details

This table is the numerical source for Figure 3. Its displayed differences are descriptive comparisons from the history-size reports. They are not pooled with the separate matched effective-arm comparison in Appendix F. In particular, the displayed DeepSeek/RunBugRun difference at $n = 4$ is 0.0010, whereas the latter report gives 0.00609. The report settings must not be assumed identical. Values displayed as ties are equal only to the available rounding precision.

Table 9: **Categorical accuracy (%) in the reported history-size sweep.** Values are reported to one decimal place. This is future-outcome prediction accuracy, not all-tests program Pass@1.

Benchmark	Backbone	$n = 1$	$n = 2$	$n = 4$	$n = 8$
RunBugRun	DeepSeek 6.7B	90.7	90.9	92.2	93.2
RunBugRun	Qwen2.5 7B	90.6	91.4	91.5	92.5
CodeARC	DeepSeek 6.7B	83.3	83.9	84.7	88.8
CodeARC	Qwen2.5 7B	80.5	81.0	83.7	86.6

F Matched-Mass Mechanism and Replication

Concentration bins. Table 3 and Figure 4(b) report the same primary DeepSeek/RunBugRun comparison at matched effective-arm settings. The bins contain 3,000 examples in total, with identical argmax predictions between estimators. Positive NLL differences favor effective mass. No bin-level uncertainty intervals were reported.

Recorded repetitions. Table 10 gives all three recorded repetitions. Each contrast matches the effective-arm parameters between predictors. NLL differences vary in sign, consistent with the conditional log-loss characterization in Appendix A.4. Initialization identifiers are defined in Appendix B.2.

Table 10: DeepSeek/RunBugRun mechanism sensitivity across recorded initializations at the effective-arm parameters. Positive NLL gain favors effective evidence over fixed mass.

Initialization	NLL gain ↑	95% CI
1701	**+0.00609**	[+0.00211,+0.01075]
1702	-0.00367	[-0.00447,-0.00287]
1703	+0.00067	[+0.00023,+0.00118]

G Derived Arithmetic and Paired-Count Interpretation

G.1 Coding illustration and supervision weights

In Figure 1, `sum(range(1, n))` gives $0, 1, 3, 6$ on inputs $1, 2, 3, 4$, while `sum(range(1, n+1))` gives $1, 3, 6, 10$. All transitions are Fix, so $r = (1, 0, 0, 0)$ for either stipulated relevance vector: $(0.25, 0.25, 0.25, 0.25)$ or $(0.97, 0.01, 0.01, 0.01)$. These are analytical choices, not measured outputs of the lexical kernel. Their masses are 4 and $1/0.9412 \approx 1.062473$. With $\alpha = 0.1$, $\lambda = 0.5$, and $v = (1, -1, 0, 0)$, Equations 5–8 give $w \approx 0.832081$ and 0.541946. Fixed mass assigns 0.832081 to both. Targets and KL coefficients are held fixed; the comparison concerns supervised coefficients, not measured generation performance.

G.2 Reported mass fractions and paired counts

For the four-observation matched comparison in Table 3, the retained pseudo-count fraction is $E(p)/4$. The three bin means therefore retain $100(1.511/4) = 37.775\%$, $100(2.521/4) = 63.025\%$, and $100(3.681/4) = 92.025\%$ of fixed mass. The main text rounds these to one decimal place. These are mass ratios; they are not ratios of training weights, measured gradient norms, or independent test counts.

For paired downstream outcomes, let N_{01} count sources that fail under `no_update` and pass under EESD, and N_{10} count the reverse. The difference in solved-source counts equals $N_{01} - N_{10}$, giving

$$\Delta_{\text{pp}} = 100\frac{N_{01} - N_{10}}{N}, \qquad N = 500.$$

Aggregate solved counts identify this net difference but not N_{01} and N_{10} separately. We retain the supplied paired bootstrap intervals rather than reconstructing them from marginal totals.

H Scope of the Empirical Evidence

The reported downstream matrix compares the complete EESD update with `no_update`. It contains no numerical results for matched equal-weight correction SFT, correctness filtering, confidence weighting, fixed-mass training, mean-utility-only weighting, or removal of the KL anchor. The comparison therefore assesses the complete update, not any component in isolation.

Appendix B specifies the scope of the reported evaluations, including randomization, uncertainty, and task-level separation.