HARNESS LEARNING ENABLES GENERALIZABLE TEST-TIME ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

A language-model agent is jointly defined by its model and its harness, the executable program that organizes model calls, tool use, and information flow. Different tasks call for different ways of organizing these operations, creating a need to adapt the harness using feedback from the task at hand. We introduce *harness learning*, which trains a proposer model to revise a solver's harness using execution information. We formulate this process as meta-learning over executable programs, with harness revisions playing the role of weight updates in gradient-based adaptation. We train the proposer through reinforcement learning using the task performance of revised harnesses as the reward. At test time, the proposer uses feedback from successive executions on a new task to refine the harness, while both models remain frozen. Experiments across reasoning and multi-hop question answering show improved revision quality and transfer to unseen tasks. Policies trained on individual revisions can continue improving harnesses over multiple rounds, while the benefits of training on revision sequences vary across settings. These findings suggest a path towards continually learning agents that turn accumulated experience into generalizable improvements.

1 INTRODUCTION

A *harness* is the executable program that organizes calls to a foundation model and interactions with tools (Yang et al., 2024; Khattab et al., 2024). It determines what the model sees, which tools it can use, and how execution proceeds. Because these choices strongly shape performance (Shinn et al., 2023; Yang et al., 2024), adapting the harness provides a way to improve how a model uses its capabilities without changing its parameters (Zhang et al., 2025b; Lou et al., 2026).

The harness design space spans choices about model calls, tool use, and intermediate outputs. Execution feedback can guide these choices, but finding effective revisions often requires repeated experimentation at test time, and revisions that work on one task may not transfer to another. We therefore ask *whether a model can learn to revise harnesses from execution feedback in a way that generalizes to new tasks*. We call this problem *harness learning*. We view harness learning as *meta-learning* over executable programs (Duan et al., 2016; Finn et al., 2017). The outer loop trains a revision policy from execution outcomes, while the inner loop uses that policy to adapt a harness from execution feedback. Training therefore learns a reusable revision procedure that can transfer to unseen tasks without updating model parameters at test time (Baxter, 2000; Andrychowicz et al., 2016).

We implement harness learning by training a model to edit executable harness code (Figure 1). Given a task description, the current harness, and an execution report, the revision model proposes a code change. We train the revision model with reinforcement learning from the performance of the resulting harness, optionally after a supervised finetuning stage. At test time, the same revision policy can be applied repeatedly, using feedback from each execution to guide the next revision.

We evaluate harness learning on reasoning and multi-hop question answering. On Reasoning Gym (Stojanovski et al., 2025), training improves revision quality on task families excluded from both supervised and reinforcement learning, and the trained 4B revision model outperforms the 35B teacher on average in single-step revision. In multi-hop question answering, a revision model trained on HotpotQA (Yang et al., 2018) transfers to MuSiQue (Trivedi et al., 2022) and 2WikiMultihopQA (Ho et al., 2020). In both settings, learned revision policies can continue improving harnesses over

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.

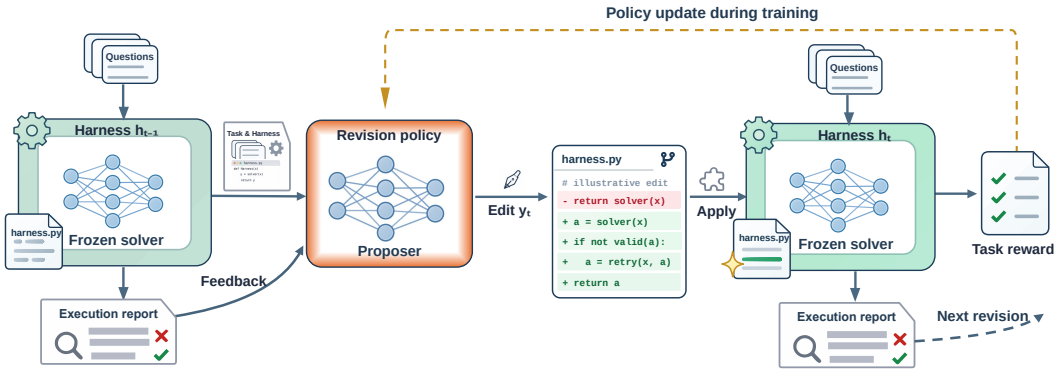


Figure 1: **Harness learning.** The proposer reads a task description, the current harness h_{t-1}, and its execution report, then generates a code edit y_t. The illustrated edit adds answer validation and a retry to a direct solver call. Executing the retained revision h_t produces a task reward and feedback for the next round. During training, rewards update the proposer while the solver remains frozen. At test time, both models remain frozen, and adaptation changes only the harness code. Figure 2 details candidate scoring and selection during training.

successive rounds of test-time adaptation. Together, these results show that harness adaptation can be learned as a transferable capability.

We make the following contributions.

Learning to adapt executable harnesses. We introduce harness learning, which trains a revision policy from execution outcomes to adapt the executable harness around a frozen solver.

Generalization to unseen tasks. We show that learned revision policies transfer to reasoning tasks and question-answering benchmarks excluded from training, including settings without teacher-generated revision demonstrations.

Iterative test-time adaptation. We show that policies trained on individual revisions can continue improving harnesses over successive rounds of test-time adaptation. We analyze how training changes revision reliability, harness structure, and the use of generated tools.

2 Related Work

Harness optimization. Automated agent design optimizes language-model pipelines and workflows (Khattab et al., 2024; Zhang et al., 2025b), with harness-specific methods using execution feedback to revise the programs that govern agent behavior (Lee et al., 2026; Lou et al., 2026). Contemporary work also trains dedicated models for harness construction and revision (Shao et al., 2026; Zhang et al., 2026a). Harness-R1 trains an editor through supervised initialization and reinforcement learning with a frozen solver, a training recipe similar to our setting. JIT-Agent learns task-conditioned harness generation, repair, and evolution. Our study frames harness revision as a learned adaptation procedure and examines whether successive rounds of execution feedback improve performance on task families excluded from training.

Meta-learning. Meta-learning uses experience across tasks to learn an initialization or an adaptation procedure (Duan et al., 2016; Finn et al., 2017; Hospedales et al., 2022). AdaptFlow learns a shared workflow initialization (Zhu et al., 2025), and The Last Harness proposes an outer loop over harness-evolution blueprints (Seong et al., 2026). Our formulation treats executable harness code as the adapted object and trains the proposer as the adaptation rule. This lets the inner loop change control flow, tool interfaces, and the sequence of model calls, with the proposer learning how to make these discrete changes from execution feedback. At test time, both proposer and solver remain frozen, and graded development feedback guides successive harness revisions. TTHE also adapts executable harnesses with frozen models, using unlabeled execution traces as feedback (Nie et al., 2026). Appendix A discusses broader adaptation methods and evaluation protocols.

3 PRELIMINARIES

3.1 HARNESS

Given a language model, a *harness* is the software layer that orchestrates the model's use throughout a task. The harness constructs the context for each model call, controls when calls occur, executes tool requests, and incorporates environmental feedback to support iterative problem solving. Harness design determines how a model interacts with its environment. For example, a coding-agent harness can expose tools for inspecting files, editing code, and running tests, then return the resulting observations to the model (Yang et al., 2024). These execution choices can be changed while keeping the model's parameters fixed.

3.2 REINFORCEMENT LEARNING WITH VERIFIABLE REWARDS

A *policy* $\pi_\theta(y \mid x)$ is an autoregressive language model that maps a context x to a token sequence $y = (y_1, \ldots, y_{|y|})$. In reinforcement learning with verifiable rewards, a programmatic grader evaluates responses or their execution outcomes. The training objective maximizes the expected reward over a prompt distribution $\mathcal{D}$,

$$\mathcal{J}(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\,\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}\big[r(x,y)\big]. \tag{1}$$

GRPO. Group relative policy optimization (Shao et al., 2024) normalizes rewards within a group of responses $\{y_i\}_{i=1}^G$ sampled from $\pi_{\theta_{\text{old}}}$ for the same context. With rewards $r_i = r(x, y_i)$, each token of y_i receives the group-normalized advantage $\hat{A}_i = (r_i - \text{mean}_j\, r_j)/(\text{std}_j\, r_j + \delta)$ and $\omega_{i,k}(\theta) = \pi_\theta(y_{i,k} \mid x, y_{i,<k})/\pi_{\theta_{\text{old}}}(y_{i,k} \mid x, y_{i,<k})$. The clipped surrogate objective is

$$\mathcal{J}(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\,\mathbb{E}_{\{y_i\}\sim\pi_{\theta_{\text{old}}}(\cdot|x)}\left[\frac{1}{\sum_{j=1}^G |y_j|}\sum_{i=1}^{G}\sum_{k=1}^{|y_i|}\min\Big(\omega_{i,k}(\theta)\hat{A}_i,\ \text{clip}(\omega_{i,k}(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_i\Big)\right] \tag{2}$$

4 HARNESS LEARNING AS META-LEARNING

Harness learning trains a proposer to adapt the executable harness of a fixed solver. The proposer learns from revision outcomes during training and uses execution feedback to revise harnesses on new tasks at test time. We first define how a harness is evaluated, then describe the adaptation loop and the procedure used to train its revision rule.

4.1 TASKS, HARNESSES, AND EXECUTION FEEDBACK

A task provides a description, a set of questions, and a grader that assigns scores in $[0, 1]$. The task is the unit of harness adaptation. In our experiments, a task is either a Reasoning Gym family, such as `maze` or `sudoku`, whose questions share a generator and a grader, or a multi-hop QA benchmark such as HotpotQA. A harness h answers questions by organizing calls to the solver, tool use, and direct computation. We keep the solver fixed and omit it from the notation.

Harness quality. Let $\text{score}(h, q)$ denote the observed grader score from executing harness h on question q. We evaluate a harness on a finite question set Q using its mean score,

$$J(h; Q) = \frac{1}{|Q|}\sum_{q\in Q}\text{score}(h, q). \tag{3}$$

The score is empirical and can vary across executions when the solver is stochastic. Execution also produces traces, which we summarize together with task outcomes in an *execution report*. Appendix F gives the report formats.

Question sets. We use Q^{fb} to construct execution reports, Q^{score} to score candidate revisions, and Q^{eval} to measure held-out performance. Evaluation questions are disjoint from all questions used for feedback or candidate selection during adaptation. Feedback and scoring sets may coincide when the evaluation protocol uses the same development questions for both roles. Round subscripts indicate the sets used at a particular revision round.

4.2 HARNESS ADAPTATION AND THE META-LEARNING OBJECTIVE

The proposer π_θ defines the learned revision rule. At test time, its parameters θ and the solver's parameters remain fixed while the harness changes. Starting from a seed harness h_0, adaptation proceeds for T rounds.

At round t, the proposer input x_t contains the task description, the current harness h_{t-1}, and a report from executing that harness on Q_t^{fb}. The input may also include selected information from earlier rounds. The proposer samples G responses, applies their edits, and retains a harness according to the selection protocol,

$$\begin{aligned} y_i &\sim \pi_\theta(\cdot \mid x_t), && i = 1, \dots, G, \\ h'_i &= \mathrm{Apply}(h_{t-1}, y_i), \\ h_t &= \mathrm{Select}\big(h_{t-1}, \{h'_i\}_{i=1}^{G}; Q_t^{\text{score}}\big)\,. \end{aligned} \tag{4}$$

The index i identifies candidates within the current round. Apply extracts code edits from the generated response and applies them to the parent harness, marking proposals invalid when their edits cannot be parsed or applied. In our structured interface, an edit identifies a span of existing code and supplies its replacement.

Select evaluates candidates on the scoring questions and applies the protocol's validity and retention rules. For example, our Reasoning Gym protocol samples one candidate per round ($G = 1$), uses the same development questions for feedback and scoring, and retains the candidate only if its development score exceeds that of the current harness (Appendix B.1). Our QA protocol samples eight candidates, scores those that fail to parse or run as zero, and advances to the highest-scoring candidate even when it scores below its parent (Appendix D.3). Executing the retained harness on the next round's feedback questions provides the report for x_{t+1}.

Learning to adapt. For a fixed proposer, the revision loop maps a seed harness h_0 to an adapted harness h_T. Harness adaptation forms the inner loop of meta-learning. The outer-loop goal is to learn proposer parameters that improve the held-out performance of the adapted harness,

$$\max_\theta \ \mathbb{E}\big[J(h_T; Q^{\text{eval}})\big]\,. \tag{5}$$

Here h_T is produced by Eq. (4) using π_θ. The expectation averages over the training-task distribution, sampled questions, proposed revisions, and stochastic harness executions. Training can draw from one or several tasks, as specified in Section 5.1.

Our training procedure uses the immediate performance of individual revisions as a surrogate for this final-harness objective. Each update assigns credit from the candidate's own outcome. Training on successive revision states changes the inputs encountered by the proposer while retaining this local objective.

4.3 HARNESS LEARNING

Training updates the proposer while keeping the solver fixed. Each revision input x is constructed from a training task, a parent harness, and an execution report. The input distribution $\mathcal{D}$ in Section 3.2 is therefore a distribution over these revision contexts. Question sets supply the examples used to build reports and evaluate candidates.

Optional supervised initialization. We can initialize the proposer with successful teacher revisions. For each collection context, a teacher receives the seed harness and its execution report and proposes edits. We execute the resulting harnesses and retain revisions that improve on the seed and pass the quality filters in Appendix B.2. Supervised fine-tuning maximizes the likelihood of the retained teacher responses, providing an initialization for learning from the outcomes of the proposer's own revisions.

Reinforcement learning from revision outcomes. For each input, we sample candidate revisions as in Eq. (4) and evaluate them on a common scoring set from the same task. During training,

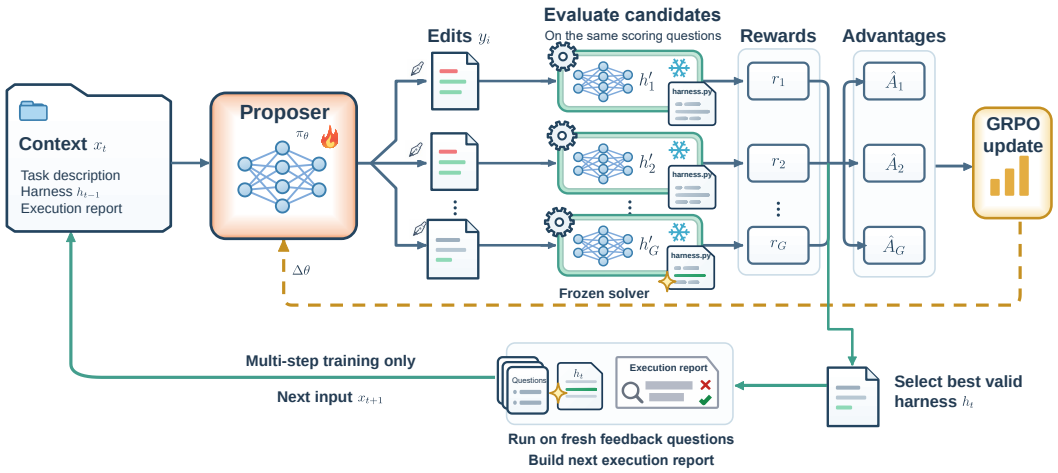


Figure 2: **Reinforcement learning for harness revision.** From input x_t, the proposer samples G responses containing edits to the parent harness h_{t-1}. Executing candidates h'_i on the same scoring questions yields rewards r_i and group-normalized advantages for proposer rollouts. When constructing successive revision states, a selected harness h_t is executed on fresh feedback questions to form the next input x_{t+1}. Feedback and scoring questions are disjoint during training so rewards measure performance beyond the feedback examples.

scoring questions are disjoint from those used to construct the input's execution report. An evaluable candidate receives reward

$$r_i = J(h'_i; Q^{\text{score}}) + v_i, \tag{6}$$

where v_i is an auxiliary reward for edit validity and successful execution. These rewards determine the group-normalized advantages used by GRPO (Section 3.2). Appendix B.3 specifies the validity terms for Reasoning Gym, where failed edits receive only accrued validity credit and explicit no-change responses are scored by executing the parent. Figure 2 illustrates the update.

Training on successive revision states. Single-step training treats each parent harness and report as an independent revision context. Training on successive revisions constructs additional contexts from selected candidate harnesses and their refreshed execution reports. Reasoning Gym refreshes these states between offline training phases, while QA generates revision sequences online. Each state uses the same immediate-reward update, so both procedures train the proposer to improve the harness presented in its current input. Section 5.1 and Appendix B.4 describe the configurations. Either trained proposer can be applied for multiple revision rounds at test time.

5 Experiments

To evaluate harness learning, we organize our experiments around three questions:

- Do learned revisions generalize to unseen tasks beyond training?
- Does successive test-time revision improve harnesses beyond a single revision?
- Does training on revision sequences improve adaptation over training on individual revisions?

We study them in two settings with separately trained proposers. Reasoning Gym tests generalization across diverse synthetic task families from a minimal seed harness that makes one solver call. Multi-hop question answering tests whether learned revisions improve an existing retrieval workflow.

5.1 Experimental setup

Table 1 lists settings. Protocols appear in Appendices B and D. *Unseen* (OOD) tasks are excluded from the proposer's supervised fine-tuning and reinforcement learning. Seed denotes the unchanged

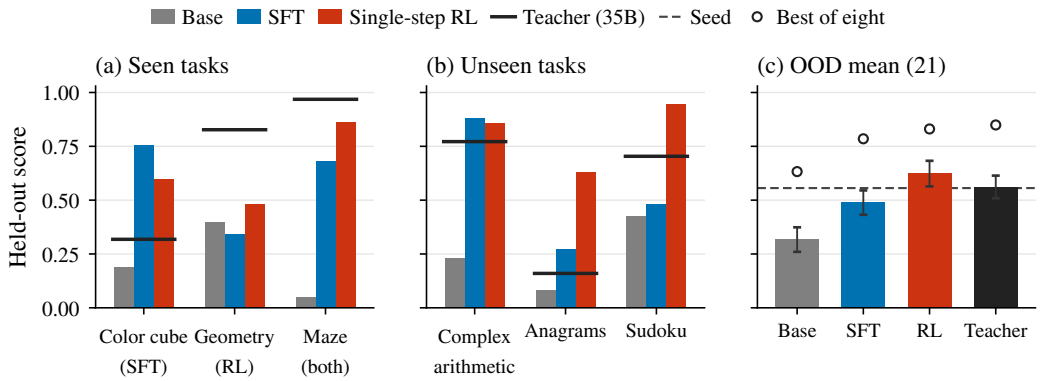


Figure 3: **Single-step harness revision.** Held-out scores on format-varied questions. (a,b) Three selected seen and three selected unseen families; bars average eight proposals and black lines denote the teacher. (c) Mean over 21 unseen families, with one standard error across families. Circles show oracle best-of-eight scores, selected on held-out scores; the dashed line marks the seed performance. Full results are shown in Table 9.

harness, while Base denotes revisions generated by the pretrained proposer. SFT, Single-step RL, and Multistep RL denote training configurations, each usable for single-step or multistep revision. For Reasoning Gym, *canonical* questions use the benchmark's standard format, while *format-varied* questions present the same tasks in alternative formats.

Table 1: **Main settings.** Oracle statistics select candidates on held-out scores.

	Reasoning Gym	Multi-hop QA
Models	Qwen3.5-4B proposer and solver	Qwen3-4B proposer, Qwen3-8B solver
Training	SFT on 35B-teacher revisions, then RL; Multistep RL continues from Single-step RL	RL from Base; Multistep RL trains on online revision runs
Tasks	21 SFT and 5 RL families; 21 unseen families	Train on HotpotQA; MuSiQue and 2WikiMultihopQA unseen
Revision	8 independent proposals on format-varied questions; 5 rounds of one proposal, kept only if development score improves	80 independent proposals; 4 runs of 10 rounds, 8 proposals per round
Reported	Held-out score; best of eight as an oracle	Held-out exact match; best of N as an oracle

5.2 Learning transferable revisions on Reasoning Gym

Generalization of individual revisions. Both training stages raise the average quality of single revisions on families excluded from training (Figure 3 , Appendix B.1). The improvement does not depend on candidate selection. After RL, the average proposal already exceeds the seed harness, while Base and SFT proposals fall below it on average.

Comparison with the teacher. On the 21 unseen families, the trained 4B proposer's revisions score above those of its 35B teacher on average (0.62 versus 0.56) under the same revision protocol. The teacher achieves higher scores under oracle best-of-eight selection. Learning from execution outcomes after supervised initialization can therefore raise average proposal quality above the teacher's. Appendix B.1 details the comparison, including the treatment of failed edits and zero-scoring harnesses.

Gains across revision rounds. On format-varied questions, Single-step RL obtains most of its five-round gain in the first revision, while SFT improves more gradually across rounds (Figure 4b). On canonical questions, the two proposers improve on similar schedules (Figure 4a). Section 6.3 discusses why later revisions add less.

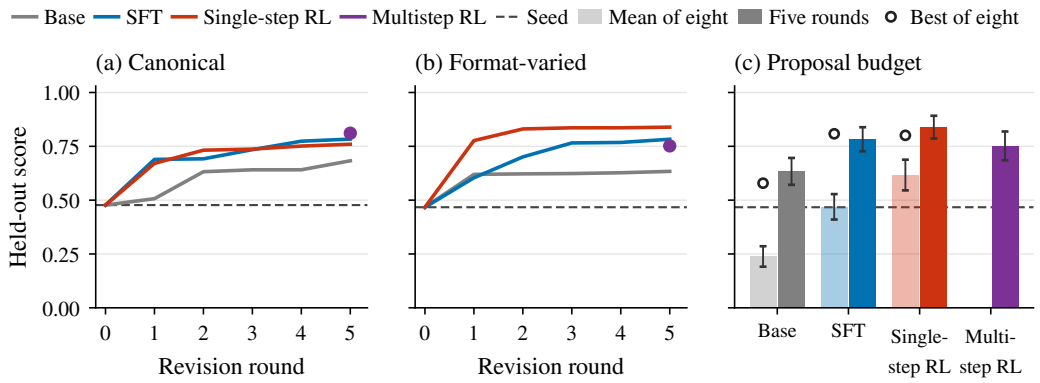


Figure 4: **Multistep harness revision on unseen families.** (a,b) Held-out scores over five revision rounds on canonical and format-varied instances, averaged over the 15 unseen families shared with the single-step comparison. Harnesses are retained by their development score, measured on separate questions used for feedback and revision selection. Multistep RL is shown at round five; dashed lines denote the seed. (c) Format-varied scores on the same families for eight independent proposals and five sequential revisions. Hollow circles denote oracle best-of-eight scores; error bars show one standard error across families.

Sequential versus independent revision. With five sequential proposals, the retained harness approaches or exceeds the oracle best of eight independent proposals and clearly exceeds their average (Figure 4c). Building on the current best harness therefore uses a small proposal budget effectively. Appendix C.2 gives in-domain and per-family results.

5.3 Adapting retrieval workflows on unseen QA benchmarks

Progress over successive revisions. Both RL proposers continue to improve beyond the first revision and finish above Base on all three benchmarks, including the two unseen ones, while Base ends below the seed on HotpotQA and MuSiQue (Figure 5a–c). Improvement also continues longer than on Reasoning Gym, with the first revision providing only one third to two thirds of the ten-round gain. Section 6.3 discusses this difference.

Sequential versus independent revision. On MuSiQue, independent revisions of the seed barely improve on it on average (Table 24). Successive revision builds substantially stronger harnesses, and on both unseen benchmarks the average final run of each RL proposer exceeds even the oracle best of 80 independent proposals (Figure 5d–e). Average improvement from the seed therefore does not fully capture a policy's ability to improve harnesses through repeated adaptation. Each run uses 80 proposals, so the comparison matches proposals per run, not total compute or feedback.

5.4 Effect of training on revision sequences

Training on revision sequences brings no consistent benefit over training on individual revisions. On Reasoning Gym, Multistep RL ends above Single-step RL on canonical questions and below it on format-varied questions (Figure 4a,b), and training on fixed later-round states also shows no gain (Appendix C.3). In QA, the two configurations reach nearly identical final scores (Figure 5a–c). In both settings, policies trained only on individual revisions already improve harnesses over successive rounds, which suggests that iterative improvement can emerge from single-revision training.

6 Analysis and Discussion

We examine how training changes the reliability and structure of proposals, and what these changes suggest about repeated revision.

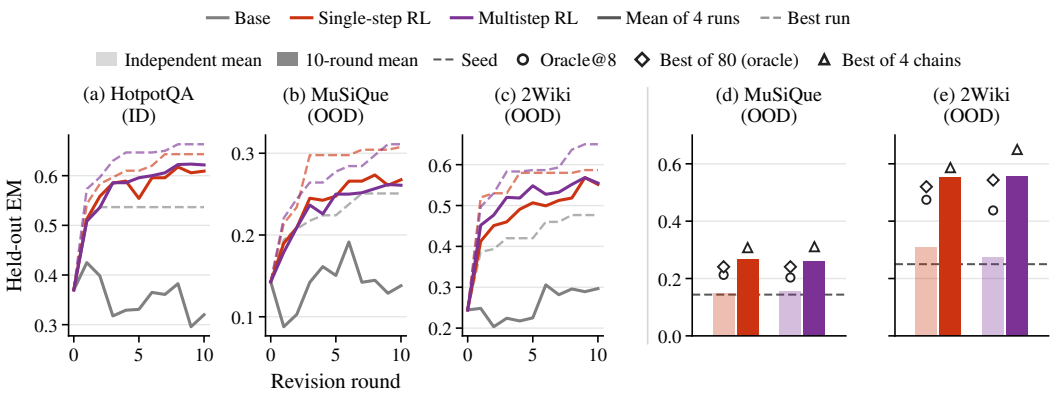


Figure 5: **Independent and sequential QA revision.** On both unseen benchmarks, mean final run scores exceed the best of 80 independent proposals. (a–c) Held-out exact match over ten rounds. Solid lines average four runs, while dashed lines show the highest score reached across all runs up to each round. (d,e) Light bars average 80 independent revisions, while dark bars average final scores from four ten-round runs, each using 80 proposals. Circles show the expected best of eight independent proposals, diamonds the best of 80, and triangles the highest score reached across all rounds and runs. These oracle statistics select on held-out scores. Horizontal dashed lines in (d,e) mark seed performance. Protocol details appear in Appendix D.4.

6.1 Training improves proposal reliability

Training reduces the fraction of proposals that fail to produce a runnable harness or score zero on the development questions (Figure 6a,d; Table 10). On unseen Reasoning Gym families, most of the reduction comes from RL, and SFT alone changes the rate little. In QA, RL from the base proposer produces a similar reduction. A trained proposer therefore gives selection more viable candidates in each round, which may help sustain progress across successive revisions.

6.2 Qualitative Analysis of Learned Harnesses

On Reasoning Gym, SFT and Single-step RL mostly generate interpreter loops, which let the solver formulate a computation as code and delegate its execution to the runtime (Figure 6b; Table 17). Interpreter loops are also the most common structure in the SFT data (48% of teacher revisions; Appendix B.2) and remain dominant after RL. Under a composition directive, helpers contribute to the answer in only a minority of the harnesses that integrate them into the solver's tool loop (Figure 6c), and composition rewards do not yield consistent gains (Appendix C.4). Unsuccessful composed harnesses often pair a correct helper algorithm with failures in question parsing, tool communication, or answer extraction (Table 21), so a generated component contributes only when these interfaces also work.

In QA, RL preserves multi-hop retrieval, while Base sometimes reduces the harness to a single solver call or removes the solver (Figure 6e). A common RL revision keeps summaries for guiding the next retrieval and passes retrieved passages directly to the answering call (Figure 7), which separates the evidence needed to formulate a query from the evidence needed to answer. Other revisions add a third retrieval hop or search again when a draft answer is absent from the retrieved passages (Figure 15). In both settings, the learned policies change how the harness organizes computation and information, and their transfer to unseen tasks suggests a reusable adaptation capability learned from execution outcomes.

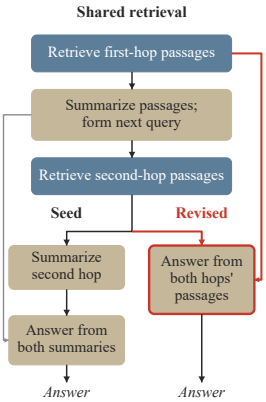


Figure 7: **An observed QA revision.** The answering call reads retrieved passages directly (red). First-hop summarization still guides retrieval.

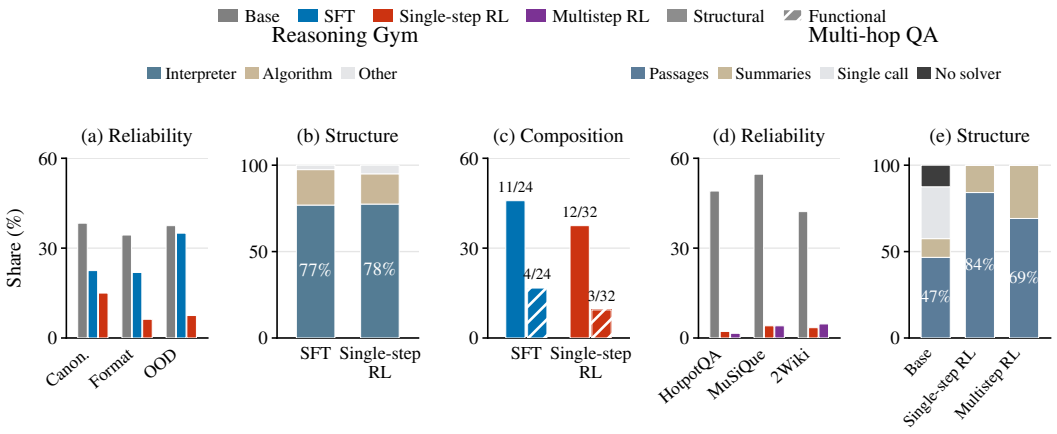


Figure 6: **Proposal reliability and harness structure in reasoning and QA.** (a,d) Failed edits or zero-scoring proposals in Reasoning Gym (15 canonical, 12 format-varied, and 5 unseen families) and QA (all 320 candidates per benchmark and method from the four ten-round runs), respectively. (b) Harness classes on five unseen reasoning families without a composition directive; "Other" combines parse-then-compute and elicitation. (c) Composition under a helper-tool directive on four unseen reasoning families. Structural composition integrates helpers into the solver's tool loop, while functional composition additionally requires a helper to contribute to the answer. Counts for (a–c) appear in Tables 10 and 17. (e) Structures of adopted QA harnesses over ten revision rounds, pooled across three benchmarks ($n = 120$ per method); "Passages" and "Summaries" denote multi-hop retrieval using retrieved passages or solver summaries.

6.3 Repeated Revision

The seed harness may explain why QA gains accumulate over more rounds than Reasoning Gym gains. On Reasoning Gym, single-step revisions commonly introduce interpreter loops. The composition experiments illustrate one obstacle to further gains: a correct helper algorithm may still fail to contribute through the surrounding parsing and tool interfaces. The QA seed already contains a retrieval workflow whose depth, evidence access, and search triggers can each be refined in later rounds. Differences in training and selection procedures prevent attributing this contrast solely to the starting harness.

Repeated revision also tests whether a policy remains useful as its own edits change the adaptation context. In QA, Single-step RL trains on seed-harness inputs yet continues to improve revised harnesses at test time (Figure 5). This suggests that its learned revision skills extend to intermediate programs and execution reports beyond those encountered in training, alongside transfer to unseen benchmarks.

Our sequence-training experiments change the harnesses and execution reports used for training while keeping rewards local to each revision (Section 4.3). They leave open whether training that assigns credit across rounds would make longer revision chains more effective. Supervised demonstrations of successive revisions, or different distributions of intermediate harnesses and feedback, could yield larger gains from multistep training.

7 Conclusion

Harness learning trains a proposer to adapt a frozen solver's executable harness using execution feedback, with both models' parameters fixed at test time. On 21 unseen Reasoning Gym families, SFT followed by RL raises mean single-step revision scores from 0.32 with the base proposer to 0.62, with the trained 4B proposer outperforming its 35B teacher on average. In multi-hop QA, RL from the base proposer supports transfer to unseen benchmarks. Policies trained on individual revisions also improve harnesses over multiple rounds, suggesting that learned revision skills remain useful as the harness changes.

AI USE STATEMENT

We used generative AI tools as coding assistants to implement, debug, and review code. Coding agents were also used to suggest and revise manuscript text under the authors' supervision. Claude Opus 5 also served as a reviewer when screening teacher-generated SFT examples (Appendix B.2). The teacher, proposer, and solver models are components of the studied method and are described in Section 5.1. The authors reviewed all AI-assisted code, analyses, and text, and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses public benchmarks (Reasoning Gym, HotpotQA, MuSiQue, and 2WikiMultihopQA) and involves no human subjects or personal data. We do not identify risks beyond those of the underlying language models.

REPRODUCIBILITY STATEMENT

Section 5.1 and Appendices B and D describe the models, data, training, and evaluation protocols. Tables 3, 5, 7 and 23 list the settings, and Appendix F gives the prompt templates. We will release code and artifacts after publishing the work.

REFERENCES

Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/hash/fb87582825f9d28a8d42c5e5e5e8b23d-Abstract.html.

Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000. doi: 10.1613/jair.731. URL https://jair.org/index.php/jair/article/view/10253.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. arXiv preprint arXiv:1611.02779, 2016. URL https://arxiv.org/abs/1611.02779.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135. PMLR, 2017. URL https://proceedings.mlr.press/v70/finn17a.html.

Prannay Hebbar, Yogendra Manawat, Samuel Verboomen, Alesia Ivanova, Selvam Palanimalai, Kunal Bhatia, and Vignesh Baskaran. SIA: Self Improving AI with Harness & Weight Updates. arXiv preprint arXiv:2605.27276, 2026. URL https://arxiv.org/abs/2605.27276.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020. doi: 10.18653/v1/2020.coling-main.580. URL https://aclanthology.org/2020.coling-main.580/.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9): 5149–5169, 2022. doi: 10.1109/TPAMI.2021.3079209. URL https://arxiv.org/abs/2004.05439.

Lisheng Huang, Chen Yang, Hao Zhou, Huatong Song, Zongchao Chen, Ran Le, Yang Song, Wayne Xin Zhao, and Tao Zhang. Evo-Bench: Can Language Models Improve Agent Harness? arXiv preprint arXiv:2608.09096, 2026. URL https://arxiv.org/abs/2608.09096.

Dongwei Jiang, Alvin Zhang, Andrew Wang, Nicholas Andrews, and Daniel Khashabi. Feedback friction: Llms struggle to fully incorporate external feedback. arXiv preprint arXiv:2506.11930, 2025. URL https://arxiv.org/abs/2506.11930.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling Declarative Language Model Calls into State-of-the-Art Pipelines. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/f1cf02ce09757f57c3b93c0db83181e0-Abstract-Conference.html.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-End Optimization of Model Harnesses. arXiv preprint arXiv:2603.28052, 2026. URL https://arxiv.org/abs/2603.28052.

Gaotang Li, Bhavana Dalvi Mishra, Zifeng Wang, Jun Yan, Yanfei Chen, Chun-Liang Li, Long T. Le, Rujun Han, George Lee, Hanghang Tong, Chen-Yu Lee, and Tomas Pfister. RubricEM: Meta-RL with Rubric-guided Policy Decomposition beyond Verifiable Rewards. arXiv preprint arXiv:2605.10899, 2026. URL https://arxiv.org/abs/2605.10899.

Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, Dakuo Wang, Xiang Zhang, Benoit Dumoulin, Cihang Xie, Yuyin Zhou, Suhang Wang, and Hanqing Lu. Harness Updating Is Not Harness Benefit: Disentangling Evolution Capabilities in Self-Evolving LLM Agents. arXiv preprint arXiv:2605.30621, 2026. URL https://arxiv.org/abs/2605.30621.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z. Pan, Alexander Du, Kurt Keutzer, Alvin Cheung, Alexandros G. Dimakis, Koushik Sen, Matei Zaharia, and Ion Stoica. EvoX: Meta-Evolution for Automated Discovery. arXiv preprint arXiv:2602.23413, 2026. URL https://arxiv.org/abs/2602.23413.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. AutoHarness: improving LLM agents by automatically synthesizing a code harness. arXiv preprint arXiv:2603.03329, 2026. URL https://arxiv.org/abs/2603.03329.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative Refinement with Self-Feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Jun Nie, Yonggang Zhang, Jun Song, Qianshu Cai, Dahai Yu, Yike Guo, Xinmei Tian, and Bo Han. TTHE: Test-time harness evolution. arXiv preprint arXiv:2607.08124, 2026. URL https://arxiv.org/abs/2607.08124.

Xuying Ning, Dongqi Fu, Tianxin Wei, Hanqing Zeng, Yuanchen Bei, Bingxuan Li, Zihao Li, Qifan Wang, Xiang Shen, Yifan Wu, Jiayi Liu, Hong Li, Yinglong Xia, Xiangjun Fan, Hanghang Tong, and Jingrui He. EvoHarness-RL: Learning self-evolving runtime harness for long-horizon LLM agents. arXiv preprint arXiv:2608.05446, 2026. URL https://arxiv.org/abs/2608.05446.

Ornith. Ornith-1.5: From Self-Scaffolding to Self-Improvement. Ornith Blog, August 2026a. URL https://ornith.ai/ornith_1_5.html.

Ornith. Ornith-1.0: Self-Scaffolding LLMs for Agentic Coding. Ornith Blog, June 2026b. URL https://ornith.ai/ornith_1_0.html.

Haebin Seong, Li Yin, Haoran Zhang, and Zhan Shi. The Last Harness You'll Ever Build. arXiv preprint arXiv:2604.21003, 2026. URL https://arxiv.org/abs/2604.21003.

Shuai Shao, Kangning Zhang, Qingyao Li, Shijian Wang, Hao Wang, Wenxiang Jiao, Yuan Lu, Yi Guo, Weiwen Liu, and Weinan Zhang. Harness-R1: Learning to Edit Executable Runtime Harnesses from Agent Failure Trajectories. arXiv preprint arXiv:2608.02276, 2026. URL https://arxiv.org/abs/2608.02276.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning Gym: Reasoning Environments for Reinforcement Learning with Verifiable Rewards. arXiv preprint arXiv:2505.24760, 2025. URL https://arxiv.org/abs/2505.24760.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/.

Wenxiao Wang, Priyatham Kattakinda, and Soheil Feizi. Do Agent Optimizers Compound? A Continual-Learning Evaluation on Terminal-Bench 2.0. arXiv preprint arXiv:2607.14004, 2026a. URL https://arxiv.org/abs/2607.14004.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. Rethinking the Evaluation of Harness Evolution for Agents. arXiv preprint arXiv:2607.12227, 2026b. URL https://arxiv.org/abs/2607.12227.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652, 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Lu Yang, Zelai Xu, Minyang Xie, Jiaxuan Gao, Zhao Shok, Yu Wang, and Yi Wu. MAGE: Meta-Reinforcement Learning for Language Agents toward Strategic Exploration and Exploitation. arXiv preprint arXiv:2603.03680, 2026. URL https://arxiv.org/abs/2603.03680.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E. Weston. Self-Rewarding Language Models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 57905–57923. PMLR, 2024. URL https://proceedings.mlr.press/v235/yuan24d.html.

Ruiqing Yue, Yu Cui, Zhuoyu Sun, Sicheng Pan, Xianhong Xue, Tingyu Li, Ting Li, Wenzhuo Zhu, Yi Chen, Yifei Liu, Baohan Huang, Zhe Cui, Haibin Zhang, and Cong Zuo. Ecdysis: Efficient and effective training of runtime harnesses for LLM agents. arXiv preprint arXiv:2609.11677, 2026. URL https://arxiv.org/abs/2609.11677.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping Reasoning With Reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488, 2022. doi: 10.52202/068431-1126. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-Taught Optimizer (STOP): Recursively Self-Improving Code Generation. In *First Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2310.02304.

Guibin Zhang, Leo Lu, Fangzhou Xie, Kang Zhu, Junhao Wang, Zhifei Xie, Zhaochen Yu, Zihang Liu, Zhongxiang Sun, Qiankun Li, Yue Liao, Heng Chang, Xiaobin Hu, Qibing Ren, Wangchunshu Zhou, Chuanrui Hu, Yafeng Deng, and Shuicheng Yan. JIT-Agent: Scaling Harness Intelligence via Just-in-Time Harness Evolution. arXiv preprint arXiv:2608.25593, 2026a. URL https://arxiv.org/abs/2608.25593.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents. arXiv preprint arXiv:2505.22954, 2025a. URL https://arxiv.org/abs/2505.22954.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating Agentic Workflow Generation. In *International Conference on Learning Representations*, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5492ecbce4439401798dcd2c90be94cd-Abstract-Conference.html.

Luan Zhang, Ruochen Zhou, Dandan Song, Zhengyu Chen, Yuhang Tian, Jun Yang, Huipeng Ma, Chenhao Li, Guangyuan Feng, Xudong Li, Yizhou Jin, and Yan Xu. HarnessCompass: Guiding Automatic Harness Evolution toward Generalizable and Effective Agent Harnesses. arXiv preprint arXiv:2608.01918, 2026b. URL https://arxiv.org/abs/2608.01918.

Runchuan Zhu, Bowen Jiang, Lingrui Mei, Fangkai Yang, Lu Wang, Haoxiang Gao, Fengshuo Bai, Pu Zhao, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. AdaptFlow: Adaptive Workflow Optimization via Meta-Learning. arXiv preprint arXiv:2508.08053, 2025. URL https://arxiv.org/abs/2508.08053.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-Adapting Language Models. arXiv preprint arXiv:2506.10943, 2025. URL https://arxiv.org/abs/2506.10943.

A Extended Related Work

Harness optimization and learned editors. Harness optimization improves executable agent programs through search or learned editing policies. DSPy optimizes modular language-model pipelines, while AFlow searches over code-based workflows (Khattab et al., 2024; Zhang et al., 2025b). Meta-Harness and AutoHarness revise harnesses from execution feedback (Lee et al., 2026; Lou et al., 2026), and Ecdysis aggregates failures across instances to guide repairs (Yue et al., 2026). EvoHarness-RL learns to manage external belief, progress, and experience state during execution (Ning et al., 2026).

Comparison with contemporary work. Harness-R1, JIT-Agent, and Ornith are contemporary efforts to learn harness construction or revision. They share our interest in learning from execution outcomes, with differences in the training signal, adaptation process, and scope of transfer evaluation (Table 2).

Harness-R1. Shao et al. (2026) train a dedicated editor with GPT-5.5 demonstrations followed by GRPO, keeping the target agent frozen. Their SFT, RL, validation, and test partitions contain disjoint instances from the same three benchmarks. They also evaluate patches on instances withheld from adaptation feedback and transfer the editor to unseen target models. Our transfer evaluation excludes entire reasoning families from both training stages and tests a HotpotQA-trained policy on MuSiQue and 2WikiMultihopQA. During Harness-R1 training, rewards come from rerunning the batch that supplied the failure feedback; our scoring questions are disjoint from the feedback questions. Their procedure generates patches without successive refinement of the same patch, whereas we study repeated revision and training on successive revision states. Supervised initialization followed by RL is shared with our Reasoning Gym setting. Our QA policy instead starts RL directly from the base proposer without teacher-generated revision demonstrations (Section 5.1).

JIT-Agent. Zhang et al. (2026a) learn task-conditioned harness generation, repair, and evolution within a four-module protocol. Training combines teacher SFT, execution-based DPO, supervised repair, and Evo-GDPO, which rewards improvements over archive incumbents while accounting for task performance, latency, and cost. Its streaming setting also accumulates execution experience to improve subsequent harnesses. Both works therefore study learned adaptation from feedback. JIT-Agent organizes adaptation around task instances and an expanding harness archive; our revision sequences update a current harness evaluated across multiple instances within a task family, with transfer measured on families or benchmarks excluded from training.

Ornith. Ornith-1.0 jointly trains a policy to generate scaffolds and solve tasks using rewards from solution rollouts (Ornith, 2026b). Ornith-1.5 extends this loop to jointly optimize task generation, harness generation, and solution rollouts (Ornith, 2026a). These contemporary releases couple improvements in scaffolding with changes to the task-solving policy. Our training updates the harness revision model while keeping the solver fixed, allowing us to evaluate harness improvements without changes to solver parameters. At test time, both models remain fixed while execution feedback guides code revision.

Table 2: **Contemporary approaches to learned harness adaptation.** Training signals and adaptation processes; the accompanying text discusses transfer evaluation.

Work	Training	Adaptation
Harness-R1	Teacher SFT followed by GRPO	Edit a frozen target agent's executable harness from failure feedback.
JIT-Agent	Teacher SFT, execution-based DPO, supervised repair, and Evo-GDPO	Generate, repair, and evolve task-conditioned harnesses using an expanding archive.
Ornith	Joint RL for scaffolding and solving; Ornith-1.5 also trains task generation	Improve harness generation together with the task-solving policy.
Harness Learning	SFT then RL on Reasoning Gym; direct RL on QA	Revise a current harness with a fixed solver; study transfer and repeated adaptation.

Meta-learning. Meta-learning uses experience across tasks to improve adaptation to new tasks (Hospedales et al., 2022). MAML learns an initialization for gradient-based adaptation (Finn et al., 2017), while RL^2 encodes a reinforcement learning algorithm in a recurrent policy (Duan et al., 2016). AdaptFlow learns a shared workflow initialization through language-guided updates (Zhu et al., 2025). The Last Harness proposes an outer loop over harness-evolution blueprints (Seong et al., 2026), and EvoX jointly evolves solutions and their search strategies (Liu et al., 2026). Meta-RL methods also learn to use interaction histories and reflections (Yang et al., 2026), or convert rubric-based judgments into reusable guidance (Li et al., 2026). Our proposer maps a harness and execution feedback to a code revision. Training uses immediate revision rewards as a greedy surrogate for final harness quality (Section 4.3).

Self-improvement. Self-improvement methods use feedback to update outputs, memory, code, or model parameters. Self-Refine revises model outputs (Madaan et al., 2023), and Reflexion stores verbal reflections for later attempts (Shinn et al., 2023). Feedback can also fail to sustain improvement (Jiang et al., 2025). STOP improves the program that proposes code changes (Zelikman et al., 2024), and the Darwin Gödel Machine evolves agent code through a candidate archive (Zhang et al., 2025a). TTHE adapts harnesses from unlabeled execution traces with fixed model parameters (Nie et al., 2026).

STaR trains on generated rationales yielding correct answers (Zelikman et al., 2022), and Self-Rewarding Language Models construct training preferences from model-generated rewards (Yuan et al., 2024). SEAL learns to generate fine-tuning data and update directives (Zweiger et al., 2025), while SIA combines harness and model-weight updates (Hebbar et al., 2026). Harness learning trains a separate proposer from revision outcomes with a fixed solver. At test time, both models remain frozen, and graded development feedback guides harness edits.

Evaluation of harness evolution. Harness evaluation accounts for solver capability, feedback, and search budgets. Wang et al. (2026b) examine benchmark overfitting under matched feedback and inference budgets, while Lin et al. (2026) distinguish generating useful updates from benefiting from them. HarnessCompass studies transfer through constrained edits and richer feedback (Zhang et al., 2026b), and Evo-Bench evaluates evolution across held-out task suites (Huang et al., 2026). Continual-learning evaluations test whether gains persist as tasks arrive (Wang et al., 2026a). Our experiments fix the solver within each benchmark, separate development feedback from held-out scoring, and report average proposal scores, best-of-eight scores, and successive-revision performance. Evaluations across benchmarks and on families excluded from both training phases test transfer of the learned adaptation procedure.

B REASONING GYM: TRAINING AND EVALUATION DETAILS

B.1 EVALUATION PROTOCOL

Reasoning Gym evaluation measures revision performance on training and unseen families under canonical and varied question formats (Table 3). Each family has disjoint development and held-out sets. We use development instances for feedback and harness selection. Held-out instances measure performance and remain hidden from the proposer.

Table 3: **Reasoning Gym evaluation settings.** Budgets apply to each task family.

Setting	Value
Development instances	25
Held-out instances	75
Independent proposals, single-step	8
Revision rounds, multistep	5
Proposals per revision round	1
Maximum generation attempts per proposal	3
Response budget per attempt (tokens)	32,768

Single-step evaluation samples independent revisions of the seed harness from a fixed report. Multistep evaluation revises the best harness retained so far using its development report. At round t, the input x_t contains the task description, the retained harness $h^\star_{t-1}$, and its report on Q^{dev}. Applying an edit $y_t \sim \pi_\theta(\cdot \mid x_t)$ produces a candidate h'_t, which we retain only if its development score improves,

$$h^\star_t = \begin{cases} h'_t, & \text{if } J(h'_t; Q^{\text{dev}}) > J(h^\star_{t-1}; Q^{\text{dev}}), \\ h^\star_{t-1}, & \text{otherwise.} \end{cases} \tag{7}$$

Methods share the seed harness and initial report within each comparison. Each subsequent round uses fresh feedback from the retained harness, whose held-out score we measure after every round. Appendices F.1 and F.2 give the revision template, execution report, and branch instruction.

We score every proposal, including zero-scoring harnesses. Failed edits and declared no-change proposals retain the parent score. The seed is excluded from proposal counts. Rejected edits receive error feedback and may be retried within the attempt budget. We report mean and best-of-eight held-out scores for single-step revision and the development-selected harness score for multistep revision (Tables 9 and 13). Edits are parsed only from the final answer. Table 10 separates failed edits from zero-scoring harnesses.

Appendix C.1 examines sensitivity to the treatment of failed edits.

Unseen families exclude all SFT and RL families. Table 9 reports every evaluated family, its training exposure, and exclusions from the main figures. The main figures exclude five unseen families that are hard to address with an algorithmic harness for both the 4B proposers and the teacher: knight_swap, letter_jumble, modulo_grid, sokoban, and ab (marked †). Over all 26 unseen families, mean scores are 0.282 for Base, 0.415 for SFT, 0.528 for Single-step RL, and 0.501 for the teacher.

The seed harness calls the solver once and extracts its answer. The fixed tool-loop baseline supplies Python execution for up to five turns and is shared across families. The teacher uses the same revision and scoring protocol and the same feedback reports.

Single-step figures include the seed baseline. Multistep curves use retained-harness scores (Eq. (7)). The multistep columns of Table 16 report the best proposal over rounds. The two scores differ by at most 0.002 in any family mean. Figure 4 compares both protocols on shared unseen families. Tables 9 and 14 give per-family results. Aggregate uncertainty is one standard error across families after averaging repeated runs within each family. Per-family shading gives the range across runs.

Scope of revision comparisons. Single-step and multistep evaluation differ in proposal budget and prompting. Independent proposals use a branch instruction and regenerate the seed report. Sequential revisions share a fixed initial report and then use feedback from retained harnesses. Our comparisons of successive and independent revision do not isolate the contribution of feedback,

because the protocols also differ in parent harness and candidate selection. Appendix C.2 details the Reasoning Gym family subsets, and Appendix D.4 describes the corresponding QA comparison and proposal budgets.

Composition experiments compare standard prompts with prompts augmented by a helper-tool directive (Appendix F.3). Structural composition integrates helpers into the solver's tool loop. The functional composition requires a helper to contribute to the answer. Table 17 reports the audit counts.

B.2 SFT CORPUS AND TRAINING

SFT uses first-round teacher revisions that improve seed performance and pass execution checks, from task families listed in Table 4.

Table 4: **Reasoning Gym training task family selection.** Task families used in either phase are excluded from the unseen-family evaluation.

Phase	Families
SFT	`advanced_geometry`, `binary_alternation`, `bitwise_arithmetic`, `boxnet`, `calendar_arithmetic`, `color_cube_rotation`, `count_bits`, `countdown`, `dice`, `fraction_simplification`, `jugs`, `knights_knaves`, `largest_island`, `maze`, `mini_sudoku`, `prime_factorization`, `products`, `simple_equations`, `syllogism`, `tower_of_hanoi`, `tsumego`
RL	`codeio`, `maze`, `rotten_oranges`, `simple_geometry`, `tower_of_hanoi`

Collection contexts contain the seed's development score, outcome counts, and failed questions with trace excerpts. The teacher independently revises Python harnesses exposing `solve(model, question)`. We execute each applied edit on the same set in a guarded sandbox. Accepted examples preserve completions verbatim, including reasoning, edits, and change manifests. Table 5 gives the collection budgets and corpus size. Each collection run samples eight first-round teacher proposals, and one to six examples are retained per run. Some collection runs add a directive asking the teacher to give the solver a Python execution tool; SFT training prompts omit this directive. Under the structural audit taxonomy (Appendix C.4), 207 of the 432 examples (48%) are interpreter loops.

Candidates must satisfy $J(h'; Q^{\text{dev}}) \geq \max\{J(h_0; Q^{\text{dev}}) + 0.08,\ 0.40\}$ and pass screens for answer echoing, duplicate code, and output-interface violations. Claude Opus 5 reviewers re-execute surviving candidates on development, held-out, and format-varied questions and inspect them for oracle access, memorization, and grader exploitation. No held-out question text enters the corpus.

Four-fold family-disjoint cross-validation over all 432 examples selects the duration of LoRA training. We then train on the full corpus for that duration and merge the adapters into the base model. Table 5 lists the optimization settings.

Limitations of supervised data. Our Reasoning Gym SFT corpus contains only first-round revisions from a single teacher and is concentrated in interpreter loops and direct algorithmic solutions. Some collection runs also use explicit directives to encourage tool use. This supervision may limit the range of revision strategies learned during SFT, particularly for improving harnesses over successive rounds. Future work could explore stronger teachers and more diverse, multistep revision demonstrations, and test whether this initialization better supports subsequent multistep RL.

B.3 SINGLE-STEP RL

Single-step RL uses a fixed pool of contexts pairing a task family, parent harness, and execution report. Feedback and reward instances share a family and use disjoint seeds. Table 6 lists contexts built from the seed and stronger parent harnesses.

The task-score term in Eq. (6) is mean grader credit on the scoring set, including partial credit. The validity reward is

$$v_i = 0.05\,\mathbf{1}[y_i \text{ parses}] + 0.10\,\mathbf{1}[h_i' \neq \perp] + 0.15\,\rho_{\text{run}}(h_i'), \tag{8}$$

Table 5: **Supervised data collection and training settings.**

Setting	Value
Data collection	
Teacher	Qwen3.6-35B-A3B
Development instances per context	25 (16 generated for `largest_island` and `tsumego`)
Failed examples in the report	Up to 8, with question text and trace excerpts
Teacher proposals per context	8
Retained examples	432
Optimization	
Adapter	LoRA, rank 32 and scaling parameter 32
Target modules	All attention and MLP projections, including gated-delta-net projections
Context length	32,768 tokens
Learning rate and schedule	10^{-4}, cosine decay, 10% warmup
Batch size and accumulation	1 example, 4 accumulation steps
Duration selection	Median best epoch from 4-fold cross-validation with disjoint families
Final training duration	Approximately 2.9 epochs

Table 6: **Composition of the single-step RL pool.** Counts of training contexts by family and parent harness.

Family	Seed parent	Stronger parent	Total
`tower_of_hanoi`	100	100	200
`codeio`	100	100	200
`rotten_oranges`	100	0	100
`simple_geometry`	100	0	100
`maze`	40	0	40
Total	440	200	640

where ρ_{run} is the fraction of scoring trials that complete. With unit task-score weight, a harness that completes every trial but answers incorrectly earns 0.30. Failed edits earn only accrued validity credit. We execute and score the parent for no-change responses.

vLLM samples revisions asynchronously. We apply each edit, execute the harness with the frozen solver, and compute task and validity rewards. Each sampled batch receives one policy update with group-normalized advantages and a loss averaged over response tokens. Table 7 gives the sampling, optimization, and execution settings.

Table 7: **Single-step RL settings.** Solver execution settings are shared between training and evaluation.

Setting	Value
Proposer sampling and optimization	
Initialization	SFT checkpoint
Candidates per context (G)	8
Contexts per update	40
Learning rate	10^{-6}
Updates per sampled batch	1
Training duration	1 epoch over 640 contexts, 16 updates
KL penalty / entropy bonus	None / none
Sampling temperature	0.6
Prompt / response budgets	12,288 / 16,384 tokens
Checkpoint saving interval	Every 2 updates
Reported checkpoint	Update 10
Harness execution	
Solver	Frozen Qwen3.5-4B, non-thinking
Solver temperature	0.7
Solver response allowance	Minimum configured budget of 32,768 tokens
Scoring instances per context	25, with partial credit
Timeout per scoring trial	300 seconds
Compute allocation	4 policy GPUs; 8 solver replicas

Single-step RL denotes the checkpoint selected at update 10; tables marked *later* use update 16. Appendix C.3 reports additional training configurations with changes to the composition directive, training pool, and reward terms.

B.4 Training on successive revision states

Reasoning Gym Multistep RL refreshes states between two offline phases, each using the immediate-reward update of Section 4.3 (Table 8). Each revision receives the full response budget.

Each state's report comes from rescoring its harness on 25 canonical development questions and lists outcome counts and up to eight failing instances. The proposer receives the report, parent, task description, and revision instruction. All candidates share the context's scoring slice.

Table 8: **Offline training on successive revision states.**

Stage	Procedure
Initialization	Start from the reported Single-step RL checkpoint (update 10).
First phase	Train for 9 updates on 62 stored parent harnesses with room for improvement.
State refresh	Advance each parent to its best sampled revision when it improves by at least 0.01, rescore all states on the same 25 development questions, and keep states at least 0.10 below their family's best score.
Second phase	Train for 15 updates on 246 states that pass the same filter, most drawn from earlier revision runs. The final checkpoint is reported as Multistep RL.

The later-round-state experiment (Appendix C.3) trains on a fixed mixture of contexts from SFT revision runs. Both methods share the inference protocol in Appendix B.1. HotpotQA generates training sequences online (Section 4.3).

B.5 TRAINING CURVES

Figure 8 shows reward and rollout outcomes for single-step RL, with one unsmoothed point per GRPO update. Appendix C.3 gives the composition-prompt and harder-task training curves.

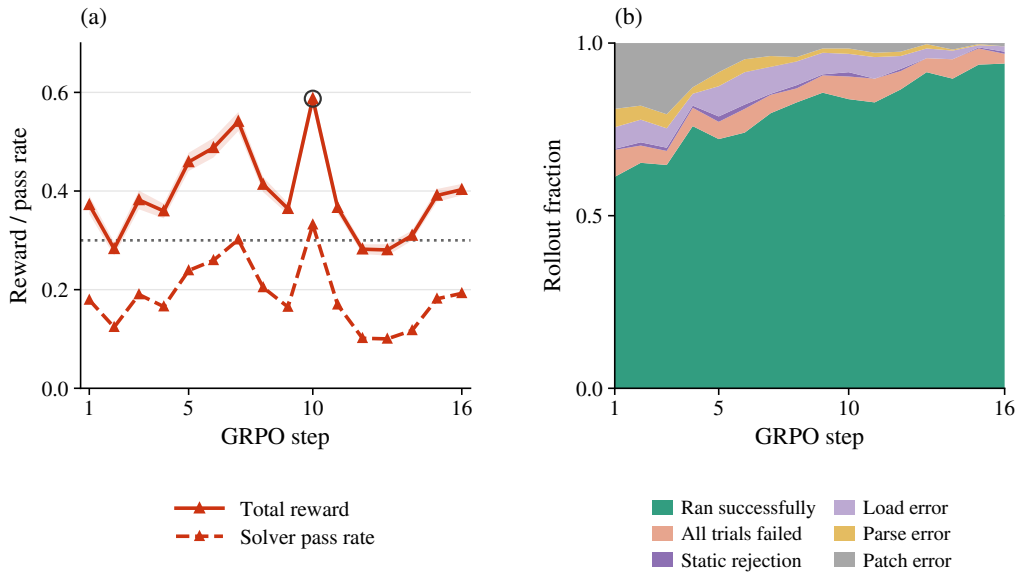


Figure 8: **Reasoning Gym RL training.** (a) Mean reward with one standard error, task score, and the validity-reward baseline. The open circle marks the reported checkpoint. (b) Rollout outcomes. Training settings are in Appendix B.3. The large variance come from prompts distribution since each batch might contains tasks from different families.

C Additional Reasoning Gym results and analyses

Methods and proposal accounting follow Appendix B. Single-step RL denotes checkpoint 10 unless specified otherwise. Table 15 defines additional training configurations, and Appendix C.4 defines composition rewards.

Captions distinguish held-out performance from development scores used for selection. Missing scores appear as $--$, and blank cells denote evaluations that were not run.

C.1 Single-step results

Figure 9 groups results by training exposure. Table 9 reports every evaluated family, including exclusions from the main comparison. Mean and best-of-eight scores measure average proposal quality and performance with candidate selection. The fixed tool-loop harness provides a common baseline. For the 27 families with teacher results in the 28-family comparison, mean scores are 0.600 for Single-step RL and 0.567 for the teacher.

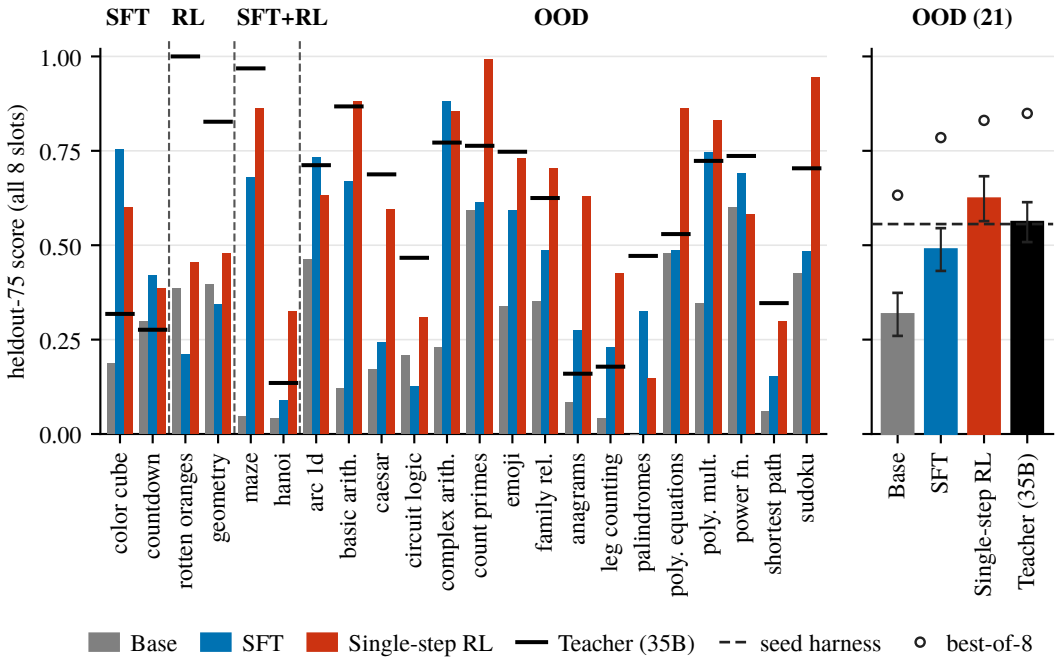


Figure 9: **Single-step revision by training exposure.** Bars show mean held-out scores; black marks denote the teacher. The OOD summary includes best-of-eight scores, the seed baseline, and one standard error across families. Table 9 gives all family results and identifies exclusions from the main comparison.

Table 10 counts failed edits and zero-scoring harnesses over all attempted proposals, including unusable edits. An applied edit can receive no task credit, so a zero score alone does not establish that the code is invalid.

Sensitivity to failed-edit scoring. Table 11 compares retaining the parent score, assigning zero, and excluding failed edits on eleven format-varied families and nineteen unseen families, which differ from the main comparison sets. Single-step RL exceeds the teacher on unseen families under all three rules. Excluding zero-scoring harnesses conditions the metric on success and can reverse this comparison, so our main results include zeros. On this nineteen-family evaluation, 30 of the teacher's 152 proposals score zero, compared with 19 for Single-step RL.

Table 9: **Single-step revision by task family.** Entries report mean / best-of-eight scores across all eight attempted proposals on 75 held-out instances per family. Single-step RL uses the selected checkpoint at update 10; *later* denotes update 16. Teacher denotes the 35B model. Families marked † are excluded from the 28-family comparison (Appendix B.1). Teacher averages cover 32 families overall and 27 in the 28-family comparison.

Family	Exposure	Tool loop	Base	SFT	Single-step RL	Single-step RL (later)	Teacher
color_cube_rotation	SFT	0.75	0.19 / 0.80	0.75 / 0.84	0.60 / 0.84	0.76 / 0.82	0.32 / 1.00
countdown	SFT	0.55	0.30 / 0.62	0.42 / 0.68	0.38 / 0.70	0.55 / 0.66	0.28 / 1.00
codeio	RL	0.32	0.24 / 0.29	0.17 / 0.29	0.27 / 0.34	0.30 / 0.36	–
rotten_oranges	RL	0.45	0.39 / 1.00	0.21 / 0.44	0.45 / 1.00	0.44 / 1.00	1.00 / 1.00
simple_geometry	RL	0.53	0.40 / 0.68	0.34 / 0.56	0.48 / 0.58	0.55 / 0.57	0.83 / 1.00
maze	SFT+RL	0.95	0.05 / 0.36	0.68 / 0.93	0.86 / 0.95	0.87 / 0.97	0.97 / 0.99
tower_of_hanoi	SFT+RL	0.24	0.04 / 0.15	0.09 / 0.28	0.32 / 0.40	0.29 / 0.39	0.14 / 0.37
ab†	OOD	0.39	0.08 / 0.47	0.28 / 0.56	0.16 / 0.43	0.33 / 0.52	0.21 / 1.00
arc_1d	OOD	0.79	0.46 / 0.81	0.73 / 0.85	0.63 / 0.81	0.76 / 0.81	0.71 / 0.81
basic_arithmetic	OOD	0.91	0.12 / 0.32	0.67 / 0.93	0.88 / 0.93	0.70 / 0.95	0.87 / 1.00
caesar_cipher	OOD	0.97	0.17 / 0.55	0.24 / 0.91	0.59 / 0.99	0.68 / 0.98	0.69 / 0.96
circuit_logic	OOD	0.13	0.21 / 0.57	0.12 / 0.25	0.31 / 0.77	0.28 / 0.77	0.47 / 0.68
complex_arithmetic	OOD	1.00	0.23 / 0.79	0.88 / 1.00	0.86 / 1.00	0.96 / 1.00	0.77 / 0.97
count_primes	OOD	0.99	0.59 / 0.69	0.61 / 0.99	0.99 / 1.00	0.80 / 1.00	0.76 / 1.00
emoji_mystery	OOD	0.66	0.34 / 1.00	0.59 / 1.00	0.73 / 1.00	0.77 / 1.00	0.75 / 1.00
family_relationships	OOD	0.75	0.35 / 0.63	0.48 / 0.75	0.70 / 0.80	0.59 / 0.81	0.62 / 0.83
group_anagrams	OOD	0.91	0.08 / 0.63	0.27 / 0.85	0.63 / 0.95	0.69 / 0.92	0.16 / 1.00
knight_swap†	OOD	0.01	0.18 / 0.53	0.00 / 0.02	0.08 / 0.49	0.05 / 0.32	0.24 / 0.53
leg_counting	OOD	0.44	0.04 / 0.12	0.23 / 0.56	0.43 / 0.57	0.55 / 0.60	0.18 / 0.37
letter_jumble†	OOD	0.29	0.38 / 0.57	0.23 / 0.31	0.34 / 0.41	0.34 / 0.44	0.48 / 0.65
mahjong_puzzle	OOD	0.49	0.34 / 0.80	0.41 / 0.72	0.48 / 0.81	0.50 / 0.68	0.47 / 0.83
modulo_grid†	OOD	0.01	0.02 / 0.07	0.02 / 0.05	0.02 / 0.04	0.03 / 0.05	0.14 / 0.92
n_queens	OOD	0.43	0.01 / 0.11	0.22 / 0.41	0.26 / 0.51	0.41 / 0.52	0.48 / 1.00
palindrome_partitioning	OOD	0.43	0.00 / 0.00	0.32 / 1.00	0.15 / 0.49	0.27 / 0.76	0.47 / 1.00
polynomial_equations	OOD	0.90	0.48 / 0.97	0.49 / 0.97	0.86 / 1.00	0.70 / 0.99	0.53 / 0.99
polynomial_multiplication	OOD	0.85	0.35 / 1.00	0.75 / 0.96	0.83 / 0.92	0.75 / 0.91	0.72 / 1.00
power_function	OOD	0.76	0.60 / 0.80	0.69 / 0.80	0.58 / 0.80	0.74 / 0.79	0.74 / 0.84
propositional_logic	OOD	0.18	0.07 / 0.23	0.14 / 0.24	0.14 / 0.23	0.13 / 0.20	0.05 / 0.23
puzzle24	OOD	0.84	0.93 / 0.95	0.95 / 0.99	0.94 / 0.99	0.90 / 0.99	0.94 / 0.97
shortest_path	OOD	0.15	0.06 / 0.31	0.15 / 0.35	0.30 / 0.88	0.28 / 0.44	0.35 / 0.35
sokoban†	OOD	0.00	0.03 / 0.20	0.01 / 0.04	0.03 / 0.12	0.03 / 0.08	0.17 / 0.68
sudoku	OOD	0.88	0.42 / 1.00	0.48 / 0.96	0.94 / 1.00	0.67 / 1.00	0.70 / 1.00
zebra_puzzles	OOD	0.97	0.81 / 1.00	0.82 / 1.00	0.85 / 0.99	0.81 / 1.00	0.34 / 1.00
mean, all 33		0.57	0.27 / 0.58	0.41 / 0.65	0.52 / 0.72	0.53 / 0.71	0.52 / 0.84
mean, 28 families		0.65	0.29 / 0.61	0.46 / 0.73	0.59 / 0.79	0.60 / 0.78	0.57 / 0.86
mean, 21 OOD families		0.69	0.32 / 0.63	0.49 / 0.78	0.62 / 0.83	0.62 / 0.81	0.56 / 0.85
mean, 27 teacher families		0.66	0.30 / 0.63	0.47 / 0.75	0.60 / 0.81	0.61 / 0.80	0.57 / 0.86

Table 10: **Proposal validity by evaluation set.** Each method attempts eight proposals per family. Failed edits yield no scorable harness (Appendix B.1). Zero-scoring harnesses have a recorded score of zero on 25 development instances. The combined rate is their sum divided by attempted proposals. Canonical, format-varied, and unseen (OOD) sets contain 15, 12, and 5 families, respectively.

Evaluation set	Method	Attempted	Failed edits	Zero-scoring harnesses	Combined rate
Canonical	Base	120	1	45	38%
	SFT	120	0	27	23%
	Single-step RL	120	0	18	15%
	Single-step RL (later)	120	0	14	12%
Format-varied	Base	96	4	29	34%
	SFT	96	0	21	22%
	Single-step RL	96	0	6	6%
	Single-step RL (later)	96	0	8	8%
OOD	Base	40	0	15	38%
	SFT	40	1	13	35%
	Single-step RL	40	0	3	8%
	Single-step RL (later)	40	0	3	8%

Table 11: **Sensitivity to failed-edit scoring.** Held-out means when failed edits retain the seed score, receive zero, or are excluded (Applied). No edit counts proposals without a revised harness. Each family has eight proposals.

	Format-varied (11 families)				Unseen (19 families)			
Method	No edit	Seed	Zero	Applied	No edit	Seed	Zero	Applied
Base	0	0.254	0.254	0.254	5	0.304	0.274	0.304
SFT	0	0.415	0.415	0.415	0	0.400	0.400	0.400
Single-step RL	0	0.576	0.576	0.576	0	0.501	0.501	0.501
Single-step RL (later)	0	0.551	0.551	0.551	1	0.526	0.519	0.525
Multistep RL					1	0.534	0.528	0.535
Composition-prompt RL (update 10)	0	0.526	0.526	0.526	1	0.496	0.491	0.493
Harder-task RL (update 14)	0	0.600	0.600	0.600	0	0.561	0.561	0.561
Teacher (35B)	0	0.551	0.551	0.551	4	0.471	0.447	0.472

All methods receive the same feedback reports. Format-varied results for Base, SFT, and Single-step RL come from the main evaluation run; other rows come from separate evaluation runs. Scoring rules are compared within each row. Blank cells denote evaluations that were not run.

C.2 MULTISTEP RESULTS

Each question format covers 24 families under the revision and promotion protocol in Appendix B.1. Twelve original families are supplemented with twelve families excluded from the main proposers' SFT and RL training. The additional families were selected partly by earlier single-step mean-per-draw performance (Table 12).

Table 12: **Selection of additional multistep evaluation families.** Eligibility uses the earlier single-step results.

Selection group	Criteria
First six families	Single-step RL outperforms SFT, which outperforms Base; the best of eight RL proposals reaches the fixed tool-loop baseline; the seed leaves room for improvement.
Remaining six	The performance-ordering requirement is relaxed to both trained proposers outperforming Base.

The canonical and format-varied sets use different family lists (Table 14). Figure 4 uses the 15 unseen families shared with the single-step comparison. We average repeated runs within each family before computing aggregate scores and standard errors. Each additional family has one run per method.

All three trained proposers finish above Base on the expanded held-out sets, with similar canonical scores and the highest format-varied score from Single-step RL (Table 13). Figure 10 separates seen and unseen families across revision rounds.

Table 13: **Held-out performance after five revisions.** Mean over 24 families per question format. Gains over Base are paired within families and reported as mean ± one standard error.

	Canonical		Format-varied	
Method	Score	Gain over Base	Score	Gain over Base
Base	0.593		0.498	
SFT	0.698	0.105 ± 0.030	0.688	0.189 ± 0.058
Single-step RL	0.699	0.106 ± 0.044	0.725	0.226 ± 0.051
Multistep RL	0.701	0.109 ± 0.038	0.649	0.150 ± 0.051

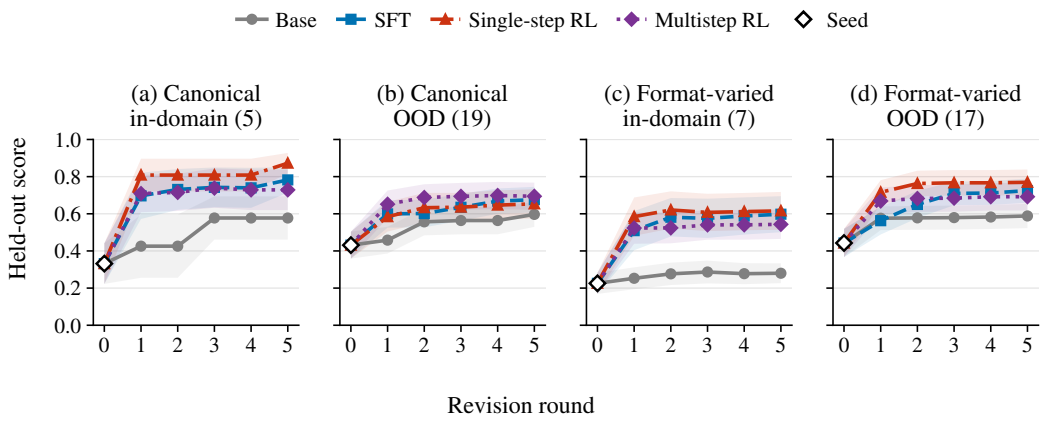


Figure 10: **Held-out performance over revision rounds.** Scores evaluate the harness retained by development-set selection, grouped by question format and training exposure. Diamonds mark the seed score averaged over runs within each question format. Each revision run retains the seed until promoting a proposal. Bands show one standard error across families after averaging repeated runs within each family.

Table 14 reports development scores after rounds 1 and 5. On `caesar_cipher`, the ordering of SFT and Single-step RL reverses between canonical and format-varied questions.

Table 14: **Multistep revision by task family.** Scores evaluate the retained harness on 25 development instances per family after rounds 1 and 5, for canonical and format-varied questions. All methods share the seed score. Exposure labels follow Table 9. Families below each dividing line form the additional evaluation set (Appendix C.2), with one run per method. Other scores average repeated runs within each family.

Family	Exposure	Seed	Base		SFT		Single-step RL		Multistep RL	
			r1	r5	r1	r5	r1	r5	r1	r5
Canonical questions (24 families)										
basic_arithmetic	OOD	0.49	0.56	0.58	0.49	0.92	0.49	0.87	0.79	0.91
bf	OOD	0.00	0.00	0.00	0.04	0.04	0.00	0.00	0.00	0.04
color_cube_rotation	SFT	0.17	0.29	0.33	0.80	0.88	0.80	0.80	0.71	0.82
count_primes	OOD	0.16	0.16	1.00	1.00	1.00	0.96	1.00	0.97	0.97
countdown	SFT	0.57	0.57	0.57	0.57	0.72	0.57	0.60	0.67	0.68
cryptarithm	OOD	0.47	0.47	0.52	0.47	0.47	0.47	0.49	0.48	0.48
maze	SFT+RL	0.08	0.08	0.64	0.96	1.00	0.64	0.88	0.84	0.97
products	SFT	0.48	1.00	1.00	0.92	1.00	1.00	1.00	1.00	1.00
propositional_logic	OOD	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.17
rotten_oranges	RL	0.20	0.20	0.44	0.20	0.56	1.00	1.00	0.48	0.55
sudoku	OOD	0.65	0.65	0.65	0.92	0.92	0.96	1.00	0.95	0.99
word_ladder	OOD	0.06	0.06	0.06	0.06	0.06	0.10	0.10	0.07	0.07
ab	OOD	0.48	0.64	0.64	0.48	0.56	0.48	0.48	0.48	0.48
arc_1d	OOD	0.72	0.72	0.80	0.80	0.80	0.72	0.72	0.84	0.88
caesar_cipher	OOD	0.00	0.00	0.64	0.16	0.88	0.00	0.07	0.88	1.00
complex_arithmetic	OOD	0.69	0.81	1.00	1.00	1.00	1.00	1.00	1.00	1.00
emoji_mystery	OOD	0.01	0.01	1.00	1.00	1.00	1.00	1.00	0.84	0.84
family_relationships	OOD	0.52	0.52	0.56	0.72	0.72	0.68	0.80	0.56	0.72
group_anagrams	OOD	0.56	0.56	0.56	0.56	0.56	0.56	0.96	0.96	0.96
leg_counting	OOD	0.52	0.52	0.52	0.56	0.56	0.52	0.52	0.52	0.52
mahjong_puzzle	OOD	0.76	0.76	0.80	0.76	0.76	0.76	0.76	0.76	0.76
n_queens	OOD	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80
palindrome_partitioning	OOD	0.16	0.16	0.16	0.24	0.60	0.20	0.60	0.16	0.88
polynomial_multiplication	OOD	0.80	0.80	0.80	0.84	0.96	0.80	1.00	0.88	1.00
mean		0.40	0.44	0.59	0.61	0.71	0.61	0.69	0.66	0.73
Format-varied questions (24 families)										
basic_arithmetic	OOD	0.43	0.43	0.43	0.80	0.93	0.84	0.92	0.58	0.86
codeio	RL	0.28	0.28	0.28	0.33	0.35	0.29	0.40	0.31	0.38
color_cube_rotation	SFT	0.25	0.25	0.41	0.45	0.84	0.70	0.74	0.70	0.74
count_primes	OOD	0.08	0.68	0.72	0.32	0.94	0.87	0.98	0.96	0.96
countdown	SFT	0.54	0.54	0.73	0.61	0.65	0.63	0.70	0.54	0.69
maze	SFT+RL	0.04	0.04	0.12	0.94	0.94	0.80	0.92	0.91	1.00
propositional_logic	OOD	0.17	0.22	0.22	0.17	0.17	0.17	0.17	0.17	0.17
rotten_oranges	RL	0.28	0.28	0.44	0.68	0.72	0.94	0.94	0.40	0.46
simple_geometry	RL	0.04	0.25	0.32	0.36	0.58	0.56	0.58	0.59	0.64
sudoku	OOD	0.65	0.65	0.89	0.92	0.98	1.00	1.00	0.96	1.00
tower_of_hanoi	SFT+RL	0.10	0.10	0.10	0.11	0.30	0.22	0.38	0.35	0.46
word_ladder	OOD	0.01	0.11	0.11	0.06	0.09	0.01	0.05	0.01	0.06
ab	OOD	0.64	0.64	0.64	0.64	0.72	0.64	0.64	0.64	0.64
arc_1d	OOD	0.72	0.72	0.84	0.72	0.76	0.72	0.72	0.72	0.84
caesar_cipher	OOD	0.00	0.59	0.63	0.88	0.88	0.96	1.00	1.00	1.00
complex_arithmetic	OOD	0.80	0.80	0.80	0.85	1.00	0.80	1.00	1.00	1.00
emoji_mystery	OOD	0.01	1.00	1.00	0.01	0.84	1.00	1.00	0.84	0.84
family_relationships	OOD	0.48	0.48	0.48	0.48	0.76	0.68	0.80	0.84	0.84
group_anagrams	OOD	0.56	0.56	0.56	0.56	0.56	0.88	0.96	0.88	0.96
leg_counting	OOD	0.52	0.52	0.52	0.52	0.56	0.52	0.72	0.56	0.56
mahjong_puzzle	OOD	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80
n_queens	OOD	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80
palindrome_partitioning	OOD	0.20	0.20	0.20	0.20	1.00	0.64	0.64	0.28	0.28
polynomial_multiplication	OOD	0.72	0.72	0.84	0.72	1.00	1.00	1.00	0.92	0.96
mean		0.38	0.49	0.54	0.54	0.72	0.69	0.74	0.66	0.71

Figure 11 brings together all family-level revision curves: canonical questions in part (a) and format-varied questions in part (b), each separating the original and additional families. Both parts use the same axes and method colors. Shading on the original format-varied set shows the range across repeated runs within a family.

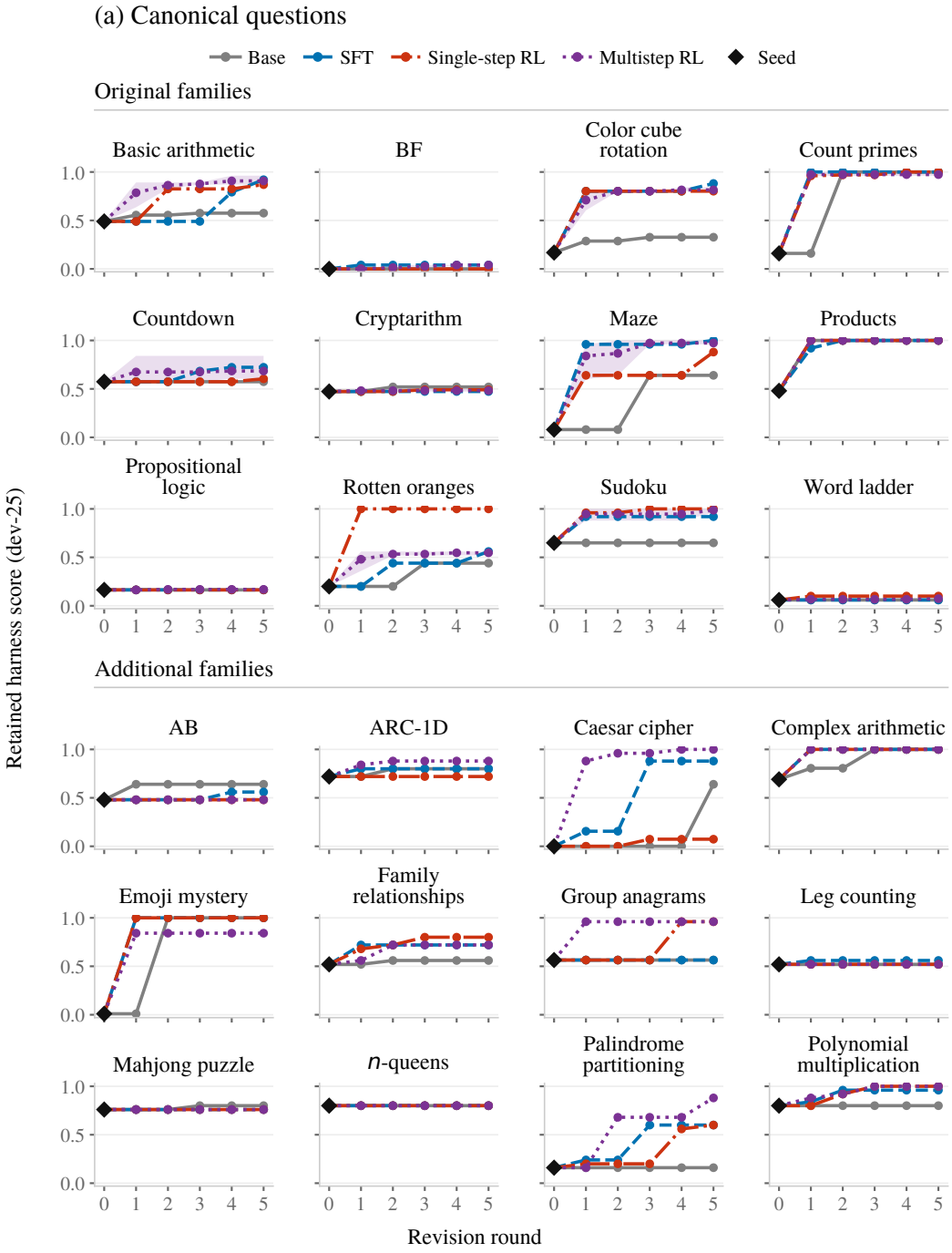


Figure 11: **Development scores across families and question formats.** (a) Canonical questions. The upper and lower blocks show the original and additional families, respectively. Curves show the harness retained after each revision; diamonds mark the seed. Families are selected as described in Appendix C.2. Part (b) continues on the next page.

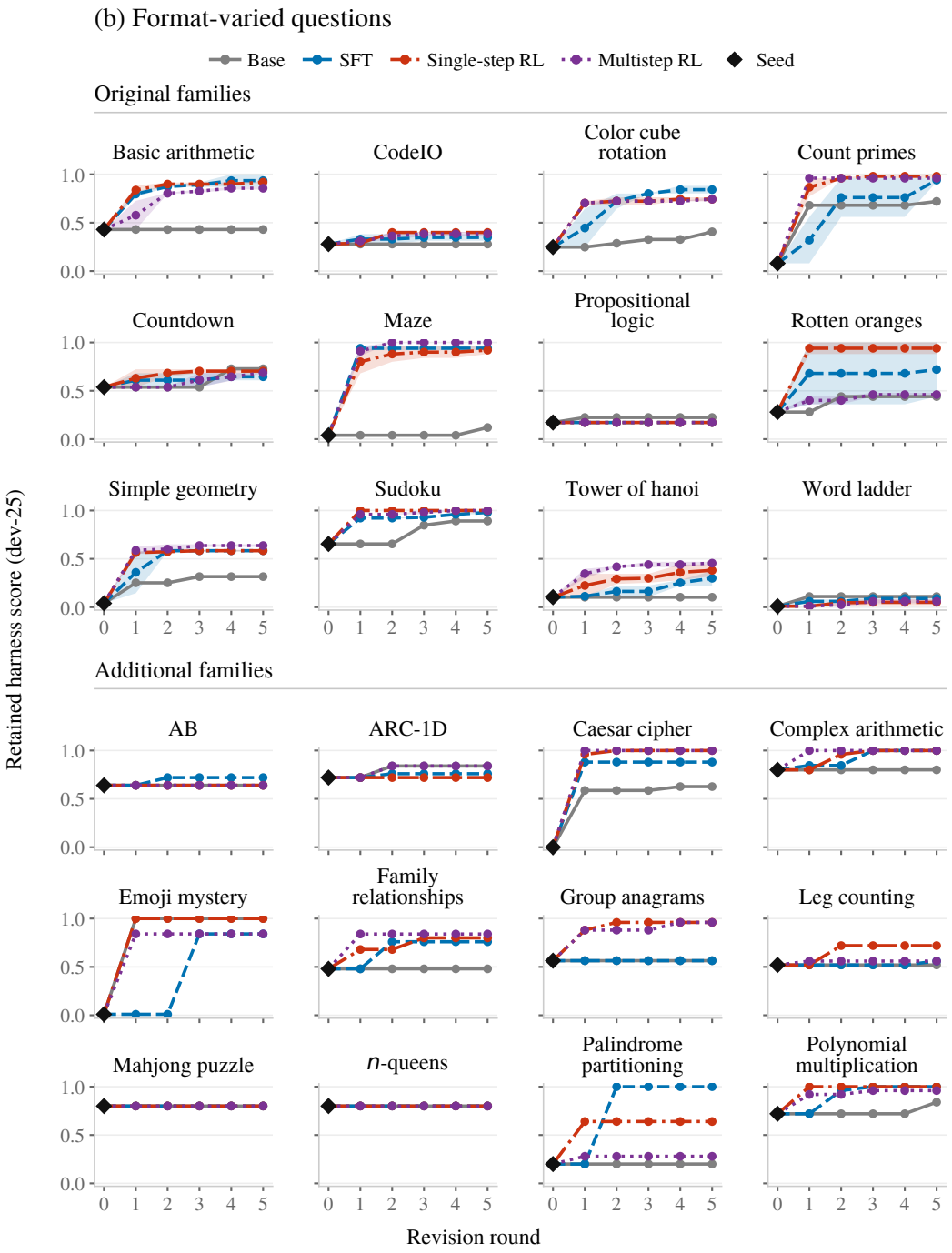


Figure 11: **Development scores across families and question formats (continued).** (b) Format-varied questions, with the same axes and method colors as part (a). Shading in the original-family block shows the minimum-to-maximum range across repeated runs. Each method has one run per additional family and question format.

C.3 TRAINING VARIANTS

Composition-prompt RL requests helper tools, while harder-task training changes the training pool and reward (Table 15). Both start from SFT and use $G = 8$, a learning rate of 10^{-6}, and no KL penalty, as in the main run. Comparisons measure the full configuration because data, prompts, and reward terms can change together.

Table 15: **Additional training configurations.** Both variants use 20 contexts per update, compared with 40 in the main RL run.

Setting	Composition-prompt RL	Harder-task RL
Training pool	600 contexts combining low-scoring cases from the single-step RL pool (Appendix B.3) with additional data; includes 100 `maze` and 100 `sokoban` contexts	694 contexts from seven families listed below
Prompt	Helper-composition directive	Canonical questions without a composition directive
Reward	Eq. (6)	Eq. (9), with revised validity terms and a structural-composition bonus
Training updates	16	14
Evaluated checkpoints	10 and 16	14

The harder-task pool contains `sokoban`, `modulo_grid`, `ab`, `leg_counting`, `rotten_oranges`, `codeio`, and `simple_geometry`. Prompt texts appear in Appendix F.3.

Composition-prompt RL produces fewer structurally composed harnesses at the later checkpoint, decreasing from 12 to 7 among 40 probe draws as the mean score increases slightly (Table 17). The score increase suggests that task reward can favor interpreter loops even when the prompt requests helper tools.

The harder-task pool contains families with room to improve over the fixed tool-loop harness and low expected scores for constant answers. Its reward includes a structural-composition bonus,

$$r_{\text{hard}} = 0.05\,\mathbf{1}[\text{loaded}] + 0.10\,\rho_{\text{run}} + J(h'; Q) + 0.15\,C(h'), \tag{9}$$

where Q is the scoring set, ρ_{run} is the fraction of trials that complete, and $C(h')$ indicates whether the harness passes the structural composition screen. No recorded rollout passes the screen. The run therefore measures the combined effect of the harder pool and revised validity terms, with no reward from the composition bonus.

Harder-task RL scores above both evaluation runs of Single-step RL on the format-varied families (0.600 versus 0.576 and 0.506; Table 16), and composition-prompt RL exceeds the main run at update 16 (0.599). Harder-task RL finishes below Single-step RL after repeated revision on both development formats. Its nineteen-family evaluation includes training families and therefore measures a mixture of seen and unseen tasks.

Later-round training states. A separate run uses a fixed pool of 465 contexts from SFT revision sequences, comprising 240 first-round contexts, 155 later-round contexts, and 70 from revision runs that remain at the seed. Training starts from SFT and follows Appendix B.3, reaching update 22 and evaluating checkpoint 16. The multistep run in Appendix B.4 refreshes states between phases.

The checkpoint reaches a round-five development score of 0.615, compared with 0.660 for Single-step RL, a paired difference of -0.045 ± 0.048 across twelve families. Gains over rounds 2 through 5 are similar, providing no evidence that the fixed later-round states improve repeated revision.

C.4 HARNESS COMPOSITION

The audit distinguishes structural composition from helpers' functional contribution to the answer. Table 17 reports both rates by evaluation group, and Figure 13 shows the full class distribution.

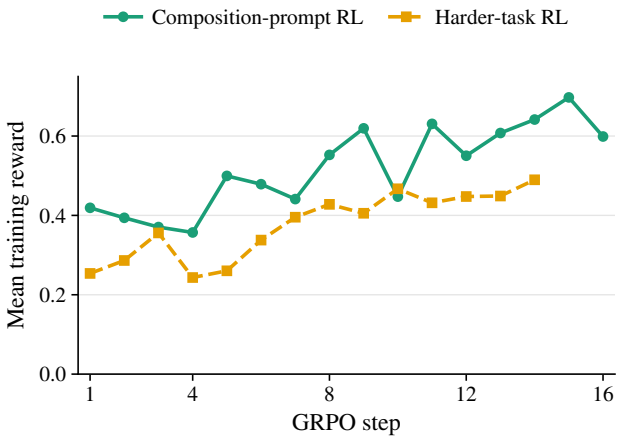


Figure 12: **Training reward for composition-prompt training and harder-task training.** Mean per update, using the different reward definitions in Eqs. (6) and (9), respectively.

Table 16: **Training variants across evaluation protocols.** Single-step scores average all eight proposals on 75 held-out instances per family. Multistep scores report the best proposal over five rounds on 25 development instances per family, using the original twelve-family sets.

	Single-step (heldout-75)		Round 5 (dev-25)	
Method	Format-varied	19-family set	Canonical	Format-varied
Base		0.304	0.496	0.398
SFT	0.463	0.400	0.645	0.625
Single-step RL	0.506	0.501	0.660	0.649
Single-step RL (later)		0.526		
Multistep RL		0.534	0.637	0.618
Composition-prompt RL (update 10)	0.526	0.496		
Composition-prompt RL (update 16)	0.599			
Harder-task RL (update 14)	0.600 / 0.595	0.561	0.640	0.643
Teacher (35B)	0.551	0.471		

Single-step format-varied results cover eleven families and use the same feedback reports as the main single-step comparison. SFT and Single-step RL values in this column come from a second evaluation run; the main run gives 0.415 and 0.576 (Table 11). Replicate scores are separated by a slash. The nineteen-family set includes families used in harder-task training. Blank cells were not evaluated.

Table 17: **Harness composition by prompt and training setting.**

Method	n	Structural	Functional	Most common class
Unseen families, without a directive				
SFT	39	0	0	Interpreter loop (30)
Single-step RL	40	0	0	Interpreter loop (31)
Unseen families, composition directive				
SFT	24	11	4	Tool loop + helper (12)
Single-step RL	32	12	3	Interpreter loop (17)
Composition-prompt RL (update 10, without directive)	32	0	0	Interpreter loop (29)
Composition-prompt RL (update 10)	32	7	5	Interpreter loop (24)
Composition-prompt RL, checkpoint probes with the directive				
Composition-prompt RL (update 10)	40	12	6	Interpreter loop (27)
Composition-prompt RL (update 16)	40	7	3	Interpreter loop (33)
Composition directive, comparison across training stages				
Base	38	0	0	Algorithm (21)
SFT	40	18	5	Tool loop + helper (19)
Single-step RL	40	15	5	Interpreter loop (22)
Guarded directive, helper with solver fallback				
SFT	40	0	0	Algorithm (40)
Single-step RL	40	0	0	Algorithm (39)
Composition-prompt RL (update 10)	40	0	0	Algorithm (40)
Composition-prompt RL (update 16)	40	0	0	Algorithm (39)

n counts audited harnesses generated from development feedback. Structural and Functional count harnesses meeting the criteria in Appendix B.1. Parentheses give counts for the most common class. Groups are separate evaluations, so counts are comparable within groups. Where specified, evaluations use the helper-composition directive. The guarded directive requests a helper with a solver fallback.

	Primary harness class						Composition	
	Interpreter loop	Loop + helper	Loop + verifier	Algorithm (no loop)	Parse then compute	Elicitation	Structural	Functional
With composition directive (5 families)								
Base (n=38)	5	39	0	55	0	0	0	0
SFT (n=40)	38	48	0	15	0	0	45	12
Single-step RL (n=40)	55	38	0	8	0	0	38	12
Composition-prompt variant (2; n=32)	50	38	0	12	0	0	38	19
Composition-prompt RL (10; n=40)	68	30	0	2	0	0	30	15
Without composition directive (5 families)								
Harder-task RL (14; n=40)	100	0	0	0	0	0	0	0
Guarded directive (5 families; level 1, 16 instances)								
Composition-prompt RL (16; n=40)	2	0	0	98	0	0	0	0
Composition-prompt RL (10; n=40)	0	0	0	100	0	0	0	0
Single-step RL (n=40)	2	0	0	98	0	0	0	0
SFT (n=40)	0	0	0	100	0	0	0	0
Format-varied questions (4 unseen families)								
Composition-prompt RL, A (10; n=32)	91	0	0	9	0	0	0	0
Composition-prompt RL, B (10; n=32)	75	22	0	3	0	0	22	16
Single-step RL, B (n=32)	53	41	0	6	0	0	38	9
SFT, B (n=24)	38	50	0	12	0	0	46	17
Format-varied questions (3 RL training families)								
Composition-prompt RL, A (10; n=24)	83	0	0	12	0	4	0	0
Composition-prompt RL, B (10; n=24)	88	4	0	8	0	0	4	0
Single-step RL, B (n=24)	58	29	0	12	0	0	25	21
SFT, B (n=22)	50	36	0	14	0	0	36	9
Hard tasks with composition directive (3 families)								
SFT (n=23)	43	35	0	22	0	0	30	0
Single-step RL (n=24)	42	42	0	17	0	0	42	0
Composition-prompt RL (10; n=24)	75	12	0	12	0	0	12	0
Teacher (35B) (n=23)	0	22	4	70	4	0	26	4
Hard tasks without composition directive (3 families)								
Harder-task RL (14; n=24)	100	0	0	0	0	0	0	0
Unseen families without composition directive (5 families)								
SFT (n=39)	77	0	0	21	3	0	0	0
Single-step RL (n=40)	78	0	0	18	0	5	0	0
Successive revision, 4 runs selected by screening (114 proposals)								
Harder-task RL (14; n=4)	25	0	0	75	0	0	0	0

Cells: %. Parentheses: training update; n = audited harnesses.

A: no directive. B: with directive. Variant: separate composition-prompt run.

Figure 13: **Generated harness classes.** Cells show percentages of audited harnesses; the final columns report structural and functional composition. A and B denote prompts without and with the composition directive.

On the eleven-family format-varied set, the composition directive increases the composition-prompt model's score but lowers Single-step RL's score (Table 18). On the five composition families, all evaluated composition-reward checkpoints score lower with the directive (Table 19).

Table 18: **Effect of composition prompting on held-out performance.** Mean over eight proposals per family on eleven format-varied families, using 75 held-out instances per family. Feedback reports are fixed across methods; all 88 proposals in each condition have recorded scores.

Method	Without directive	With directive
Single-step RL	0.506	0.461
Composition-prompt RL (update 10)	0.526	0.559

A guarded directive requesting a deterministic helper with a solver fallback produces direct algorithmic solutions in 158 of 160 draws. No draw meets the structural or functional composition criteria (Table 17). Directive versions are specified in Appendix F.3.

Composition rewards. We compare tool-use and combined-reward variants, both initialized from SFT and trained on the composition-prompt pool with the task-specific helper directive. Each checkpoint is probed on five families with and without the directive, requesting eight proposals per family. A separate twelve-family evaluation measures repeated revision.

The added reward combines structural composition with a proxy for successful tool use. Let $F(h')$ average each scoring trial's task score multiplied by an indicator that a tool result reaches the solver, with $F(h') = 0$ unless the structural screen passes. The reward is

$$r_{\text{comp}}(h, y) = r(h, y) + w_{\text{f}} F(h') + w_{\text{s}} C(h'), \tag{10}$$

where $C(h')$ is the structural indicator in Eq. (9). The tool-use reward uses $(w_{\text{f}}, w_{\text{s}}) = (0.3, 0)$; the combined reward uses $(1.0, 0.15)$, changing both weights. The audit assesses the helper's contribution to the answer separately from the reward proxy.

Under the tool-use reward, composed tool loops decline from 25 of 40 draws at initialization to one at checkpoint 14. The combined reward retains more composed loops (Table 19), while its round-five development scores remain below Single-step RL (Table 20).

Table 19: **Composition-reward probes.** Five families evaluated with and without the composition directive. Counts show the fraction of completed draws; scores use 25 development instances per family.

		With directive				Without directive	
Method	Update	Helpers	Tool loops	Loop score	All draws	Interpreter loops	All draws
SFT (init.)		32/40	25/40	0.318	0.362		
Single-step RL	10	30/40	17/40	0.469	0.493		
Tool-use reward	12	24/40	5/40	0.385	0.393	17/24	0.573
	14	19/40	1/40	0.000	0.457	36/40	0.712
Combined reward	6	30/40	17/40	0.491	0.402	36/40	0.751
	12	27/40	16/40	0.411	0.496	39/40	0.796
	20	25/40	19/40	0.485	0.509	34/39	0.690

Helpers counts proposals that use a computational helper. *Tool loops* counts proposals that integrate such helpers into the solver's tool loop. *Loop score* averages scores over those proposals, and *All draws* averages over all completed proposals. Blank cells indicate unevaluated settings.

The combined reward increases alongside execution validity and structural composition (Figure 14). Helper outputs reaching the solver in successful trials remain less common, and composed harnesses score below interpreter loops sampled during the same updates.

The checkpoint-12 inspection identifies parsing, tool communication, and answer extraction failures around correct helpers, including breadth-first search for `maze`, backtracking for `sudoku`, and a primality sieve for `count_primes` (Table 21).

Table 20: **Repeated revision after composition-reward training.** Round-five development scores on twelve canonical families, using the selection protocol in Appendix B.1 and 25 development instances per family.

Method	Checkpoint	Round-five score
SFT	Initialization	0.645
Single-step RL	10	0.660
Tool-use reward	14	0.624
Combined reward	12	0.624
Combined reward	20	0.630

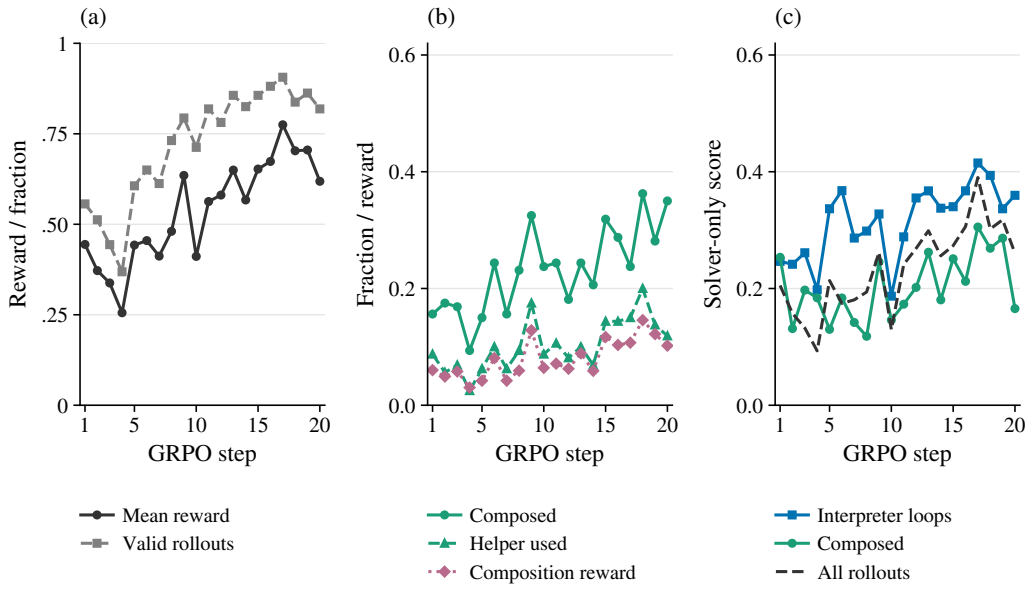


Figure 14: **Training with combined composition rewards.** (a) Reward and execution validity. (b) Structural composition, helper use, and composition reward. (c) Task scores by harness class, excluding reward bonuses.

Table 21: **Observed failures in composed harnesses.** Examples from the checkpoint-12 inspection.

Component	Observed failure	Consequence
Question parsing	The parser misses the goal marker.	Questions are routed to a fallback model call.
Answer extraction	The harness returns an unsolved grid.	The submitted answer is incomplete.
Tool communication	A tool message is malformed.	The tool interaction cannot execute.

D MULTI-HOP QUESTION ANSWERING: TRAINING AND EVALUATION DETAILS

D.1 DATASETS AND RETRIEVAL

HotpotQA provides in-domain evaluation using 5.23 million Wikipedia abstracts from its full-wiki corpus. We sample disjoint sets of 2,000 training, 500 development, and 300 held-out test questions from its training split.

MuSiQue and 2WikiMultihopQA provide out-of-domain evaluation. We pool all supplied support and distractor paragraphs from their development splits (20 per question for MuSiQue-Ans and 10 for 2WikiMultihopQA) and deduplicate them by title and text. We partition each development split into disjoint development and held-out test sets, stratified by hop count for MuSiQue and question type for 2WikiMultihopQA. Table 22 lists the corpora and question counts. Development and test questions are disjoint from each other and from the 2,000-question training pool.

HotpotQA answers are compared by normalized exact match (lower-casing and removal of punctuation, articles, and extra whitespace). On MuSiQue and 2WikiMultihopQA, a prediction is also correct if it matches one of the gold answer's listed aliases, following the official 2WikiMultihopQA script.

Table 22: **Multi-hop QA evaluation data.** Retrieval corpora and question pools per benchmark.

Benchmark	Retrieval corpus	Dev pool	Test	Split source
HotpotQA	5.23M Wikipedia abstracts (full-wiki)	500	300	train split, disjoint subsets
MuSiQue	21,100 paragraphs pooled from the dev questions	500	299	dev split, hop-stratified
2WikiMultihopQA	54,957 paragraphs pooled from the dev questions	500	300	dev split, type-stratified

D.2 RL TRAINING

On HotpotQA, RL starts from the base proposer without an SFT stage and generates its training contexts online. For each context, we run the parent harness with traces on six feedback questions Q^{fb}. The report lists the solver and retrieval calls, the final answer, whether it matches the gold answer, and which gold supporting documents were retrieved or missed. Each context also fixes 64 scoring questions Q^{score} on which all $G = 8$ candidates are scored. Both sets come from the 2,000-question training pool, and each context's draw is seeded by its index. The proposer receives the parent harness, the report, and the revision instruction, and answers with a short diagnosis followed by one Python code block, without mutation hints.

The reward is exact match on the scoring questions plus a validity bonus,

$$r_i = \begin{cases} J(h_i'; Q^{\text{score}}) + 0.1 & h_i' \neq \bot \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

where $h_i' \neq \bot$ requires that the response contains a code block that parses, defines `run`, has at most 200 lines, and completes the scoring run. Questions on which the harness raises an exception or exceeds its call budgets count as incorrect inside J, so there is no separate partial-completion term.

Single-step RL. Every context uses the seed harness as parent with its own feedback and scoring draw. The pool holds 320 contexts, consumed once in 80 updates of four contexts each. The model reported as Single-step RL is the final update.

Multistep RL. Parents come from 32 revision runs advanced by the evolving policy, providing training states from successive revisions. Each run starts at the seed harness, and contexts are ordered by run, so each run is visited once every eight updates. When a run is revisited, the candidate from its previous context with the highest scoring exact match becomes the new parent, the same best-of-eight rule used at inference; the run keeps its parent if none of the eight candidates executes. Ten

visits per run give 320 contexts and 80 updates per phase. The two policies share every other setting (Table 23), so the comparison isolates the source of the parent harness.

Training observations. In the reference training run, all 288 parent updates succeeded and no run collapsed. The mean parent score rises from 0.40 at the seed to 0.44 by depth four and then levels off. The committed child scores above its parent at every depth after the first.

Table 23: **RL settings on multi-hop QA.** Shared by Single-step and Multistep RL unless noted. Solver execution settings are shared between training and evaluation.

Setting	Value
Contexts	
Parent harness	Seed (Single-step); state of a revision run advanced by the policy (Multistep)
Feedback / scoring questions	6 / 64, disjoint, from a 2,000-question training pool
Revision runs × depth (Multistep)	32 × 10; commit by best-of-8 scoring exact match
Contexts per phase	320
Proposer sampling and optimization	
Initialization	Base Qwen3-4B, thinking enabled (no SFT)
Candidates per context (G)	8
Contexts per update	4
Learning rate	10^{-6}
Updates per sampled batch	1 (clip ratio 0.2, token-mean loss)
Training duration	Single-step: 80 updates; Multistep: 80 updates
KL penalty / entropy bonus	KL-to-reference loss, coefficient 10^{-3} / none
Sampling temperature, top-p	1.0, 1.0
Prompt / response budgets	10,240 / 8,192 tokens
Reported checkpoint	Single-step: update 80; Multistep: update 80
Harness execution	
Solver	Frozen Qwen3-8B, non-thinking
Solver temperature, top-p, top-k	0.6, 0.95, 20
Solver response allowance	2,048 tokens (16,384-token context)
Retrieval	BM25 over 5.23M Wikipedia abstracts
Call budgets per question	12 solver calls, 12 retrievals
Timeout	120 s per solver call; 900 s per scoring run
Wall time	Single-step: 3.3 h; Multistep: 2.6 h

D.3 Evaluation by successive revision

All three benchmarks use the same two-hop retrieve-and-summarize seed. Each proposer performs four independent revision runs of $T = 10$ rounds from this seed. In each round, each run draws six feedback questions from the development pool and executes its current harness on them with traces. The report lists traces, answer correctness, gold answers, and retrieved or missed supporting documents (Appendix F.2). Sixty-four scoring questions from the same pool, disjoint from all feedback questions of the round, are shared by the four runs. Development questions can recur across rounds, while feedback and scoring questions remain disjoint within each round. Each run samples $G = 8$ revisions and advances to the candidate with the highest scoring exact match, even when it scores below its parent. Reasoning Gym applies a promotion gate (Appendix B.1).

A revision whose code does not parse, does not define `run`, exceeds 200 lines, or fails its scoring run scores zero, and a run keeps its harness only if all eight revisions fail. After each round, the four harnesses are scored on the full test set. We report the mean over the four runs. The dashed curves show the highest held-out score reached across all four runs up to each round (Figure 5a–c). Random draws are seeded per round, so every proposer, including Base, sees the same feedback and scoring questions in the same round.

Sampling and execution. The proposer samples with temperature 1.0 and thinking enabled, with a response budget of 10,240 tokens. Base is the untrained Qwen3-4B under the same settings. Harnesses execute with the solver settings of Table 23. A scoring run is limited to 600 seconds and

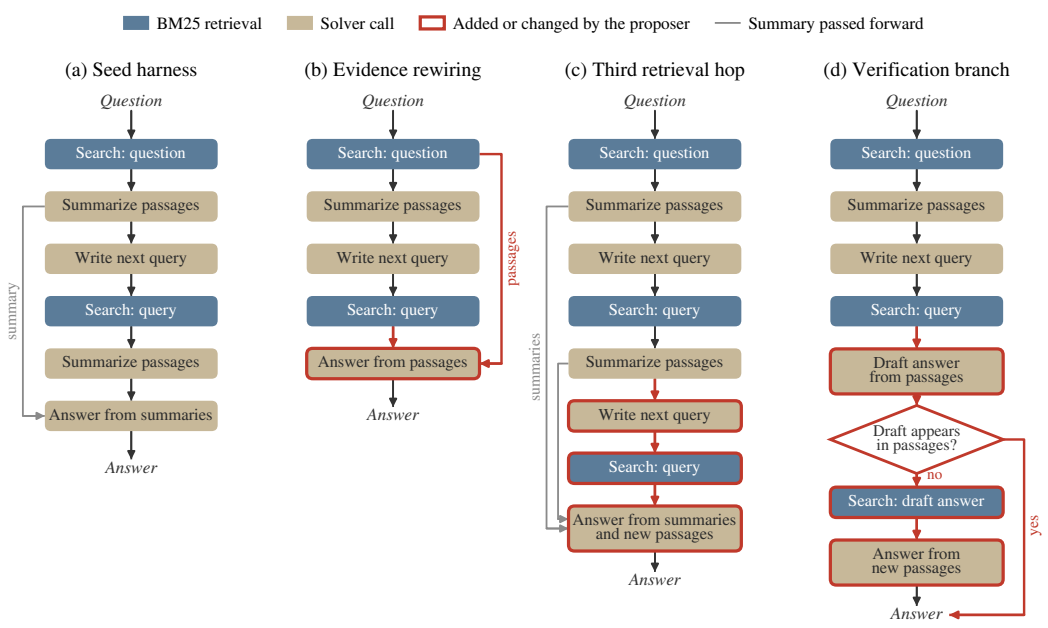


Figure 15: Seed QA harness and three edits from adopted RL harnesses (abridged). Red outlines mark additions and changes. (a) The seed retrieves twice and answers from solver summaries. (b) The answering call reads both hops' retrieved passages. (c) A third retrieval hop precedes answering. (d) A draft answer is accepted only if it appears in the retrieved passages; otherwise, the draft becomes a retrieval query and the solver answers from the new passages.

a test run to 1,800 seconds. The solver has a budget of 12 calls, and every retrieval returns at most 20 passages.

D.4 Independent single-step revision

Proposal generation and scoring. For each dataset and RL checkpoint, we generate 80 independent proposals from the seed harness, using the round-one reports of the four revision runs and 20 samples per report. This matches the 80 proposals that one revision run consumes in ten rounds. Each proposal receives the seed code and its execution feedback, produces one revised harness, and is scored by exact match on the held-out test set listed in Table 22. Edits that fail to parse or compile receive zero. Identical programs share one execution, and all 80 proposals contribute to the reported statistics.

Mean performance and oracle selection. Table 24 reports the unselected mean, the fraction strictly exceeding the seed, and the failed-edit rate. Oracle@8 is the expected maximum held-out score among eight proposals sampled uniformly without replacement from the 80. Best-of-80 is the maximum over the entire pool. Both oracle statistics use held-out scores to identify candidates and describe the quality of the generated proposals, without providing a selection rule for deployment.

Comparison with successive revision. Figure 5 compares these scores with the mean final score of four ten-round revision runs, whose selection uses only the scoring questions. Triangles show the highest held-out score reached across all ten rounds and four runs. Each run samples 80 candidates, matching the independent pool's proposal count, while the four-run mean and best-run score use 320 proposals in total. The best-run score is therefore not a matched-budget comparison with the best of 80 independent revisions. The evaluations distribute feedback differently: independent proposals share four seed reports, and successive revision obtains feedback on intermediate harnesses. Seed performance is measured separately for the independent and successive evaluations; horizontal dashed lines in the bar panels show the independent-evaluation baseline.

Table 24: **Independent single-step QA revision.** Each row summarizes 80 proposals. Oracle@8 and best-of-80 use held-out scores for selection.

Dataset	Training	Seed	Mean	Oracle@8	Best-of-80	>seed	Failed
HotpotQA	Single-step RL	0.3900	0.4105	0.5399	0.5800	71.3%	1.3%
	Multistep RL	0.3900	0.4131	0.5209	0.5833	77.5%	1.3%
MuSiQue	Single-step RL	0.1438	0.1467	0.2126	0.2408	58.8%	5.0%
	Multistep RL	0.1438	0.1551	0.2034	0.2408	67.5%	2.5%
2WikiMultihopQA	Single-step RL	0.2500	0.3090	0.4746	0.5200	67.5%	1.3%
	Multistep RL	0.2500	0.2748	0.4374	0.5433	58.8%	0.0%

E STRUCTURED AND CODING-AGENT PROPOSERS

We conduct an exploratory comparison of structured edits and an interactive Pi coding-agent session (Table 25 and Figure 16). The structured proposer receives the parent source and an execution report, then returns SEARCH/REPLACE edits and a change manifest in one model call, with retries for invalid outputs. Pi inspects files, edits the harness, and invokes tools.

Both interfaces use Qwen3.5-35B-A3B as proposer and frozen solver on Reasoning Gym Mix, Boxnet, and Sokoban. The main Reasoning Gym experiments instead use a 4B solver. Proposers use thinking mode; the solver uses non-thinking mode at temperature 0.7. Within this comparison, all runs start from the same seed program, evaluated separately for each run, and retain the highest-scoring harness, including the seed, as the next parent.

Table 25: **Development accuracy (%) with Pi and structured proposers.**

Benchmark	Proposer	Seed	Best ↑
Reasoning Gym Mix	Pi	80.0	92.0
	Structured	84.0	92.0
Boxnet	Pi	52.0	64.0
	Structured	56.0	72.0[a]
Sokoban	Pi	88.0	88.0[b]
	Structured	80.0	88.0

25 development instances per benchmark; one revision run per interface, five rounds, one candidate per round. [a]The selected candidate is byte-identical to the seed; the increase reflects re-evaluation. [b]Pi retains the original seed.

The results show that structured edits can support harness revision through a simple model-call interface. This comparison uses one run per benchmark on the development questions used for selection. Token and tool-call budgets, code snapshots, and concurrency settings differ, and independent seed evaluations yield different scores. It is separate from the trained 4B evaluations and does not establish a performance or efficiency advantage for either interface.

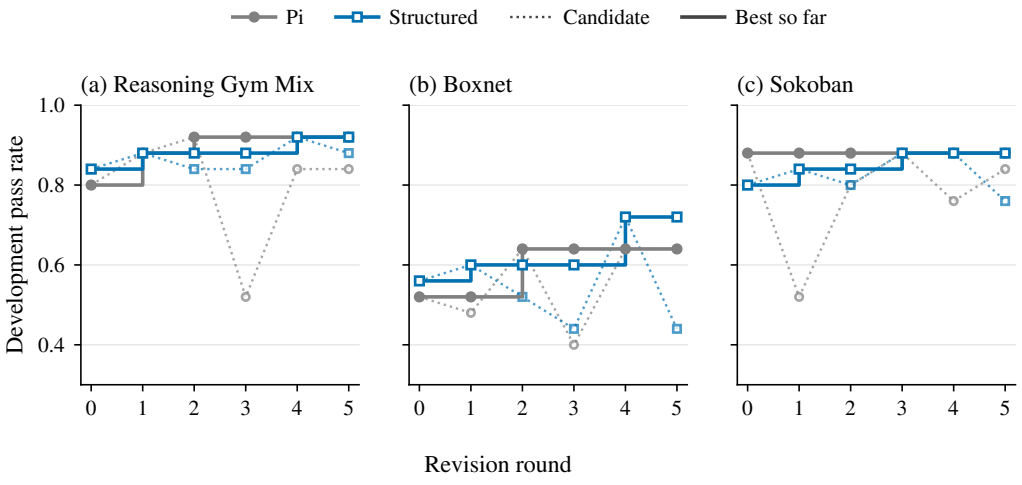


Figure 16: **Development performance over revision rounds.** Dotted curves show candidate scores; solid steps show best-so-far scores. Each interface has one revision run per benchmark.

F PROMPT TEMPLATES

F.1 STRUCTURED HARNESS REVISION

System instruction (abridged)

```
Instructions:
You are an expert Python developer improving an LLM evaluation harness by
    emitting SEARCH/REPLACE edits against the parent harness source.
Evidence discipline:
- The sole optimization target is the dev-split score of the evolved harness.
[...]
- Before any change, have all four: failure evidence (which tasks, what went
    wrong), a root cause (why), a targeted fix, and a predicted impact (tasks
     expected to flip, tasks at risk).
- Preserve working parts of the parent harness, make one targeted mechanism
    change per candidate.
[...]
- Never hard-code task-specific answers or task IDs; do not predict fixes for
     blacklisted (§7) tasks.
Emit your edits as one or more blocks in exactly this format (the <<<<<<<
    SEARCH, =======, and >>>>>>> REPLACE marker lines are literal):
<<<<<<< SEARCH
[lines copied verbatim from the parent harness, including indentation]
=======
[the replacement lines]
>>>>>>> REPLACE
[...]
```

User input

```
# Task Family
<TASK_FAMILY>
# Task Config
<task_config>
```json
<TASK_CONFIG_JSON>
```
</task_config>
Benchmark Steering Guide
<steering_guide>
<BENCHMARK_STEERING_GUIDE>
</steering_guide>
Parent harness id: <PARENT_ID> (score: <PARENT_SCORE_4DP>)
Parent harness source (the file your SEARCH blocks must match)
<parent_harness>
```python
<PARENT_HARNESS_SOURCE>
```
</parent_harness>
Evolution Evidence Report
<evidence_report>
<EXECUTION_EVIDENCE_REPORT>
</evidence_report>
<PROPOSAL_BRANCH_INSTRUCTION_IF_APPLICABLE>
<REQUIRED_CHANGE_MANIFEST_INSTRUCTION>
Emit your SEARCH/REPLACE blocks now, then the ===CHANGE_MANIFEST=== section.
    Each SEARCH must appear verbatim in the parent harness above.
Remember: keep the parent harness's entry-point contract intact and make one
    evidence-grounded mechanism change.
```

The manifest instruction below replaces <REQUIRED_CHANGE_MANIFEST_INSTRUCTION>. A rejected response receives the parsing or validation errors and a request to retry in the same format.

Change manifest

```
# Required Change Manifest
After your edits, emit this line exactly:
===CHANGE_MANIFEST===
followed by one JSON object:
{
  "changes": [
    {
      "id": "chg-1",
      "type": "new|improvement|rollback",
      "description": "what was changed and why",
      "failure_pattern": "the failure class this addresses (from the evidence
     report)",
      "predicted_fixes": ["task ids you expect to flip fail -> pass"],
      "risk_tasks": ["task ids this edit could regress pass -> fail"]
    }
  ]
}
- Emit the JSON object bare: no fences around it, no text after the closing
    brace.
- Use task ids exactly as they appear in the evidence report.
- The manifest is scored mechanically next round: predicted_fixes against
    observed fail->pass flips, risk_tasks against pass->fail regressions.
    Honest, specific predictions beat broad ones.
- Do not predict fixes for tasks on the unstable-task blacklist.
- An explicit no-op declares "changes": [] and makes no edit. A non-empty
    changes list with an unchanged file, or an edit with an empty changes
    list, is rejected.
```

F.2 Execution feedback

The execution report summarizes development-set outcomes and revision history in nine sections. Missing records appear as `n/a`; the table lists each section's contents.

Report section	Contents
1. Overview	Parent identifier and score, round, lineage, and score variation.
2. Task Classification	Instance outcomes, including partial credit and execution failures.
3. Cross-Iteration Flip Matrix	Fixed, regressed, stable, and uncomparable instances.
4. Failure Digest	Solver behavior and up to eight failed or partially correct instances, with traces capped at 2,000 characters.
5. Score Trend	Per-round and cumulative best scores.
6. Best-Ever Pointer	Identifier, score, and source of the best harness.
7. Unstable-Task Blacklist	Instances with repeatedly changing pass/fail outcomes.
8. Attribution Table	Observed outcomes of the parent's declared edits.
9. Lineage Edit History	The five most recent edits and a cumulative diff from the seed.

Independent proposals receive the same parent harness and report, with the following branch instruction. Single-proposal revision omits the instruction and uses the retained harness's updated report (Appendix B.1).

Independent proposals

```
# Proposal Branch
You are generating proposal <INDEX> of <COUNT>. Sibling proposals
are generated independently from the same evidence. Rank the failure
mechanisms by expected impact and target the mechanism ranked #<INDEX>,
so branches stay materially distinct.
```

The training and QA feedback descriptions are schematic summaries of the reported protocols. Successive-state training supplies up to K passing and K failing examples, each with the question, harness answer, expected answer, and outcome (Appendix B.4). Multi-hop QA supplies six

development questions per revision run, with execution traces, answer correctness, gold answers, and retrieved or missed supporting documents (Appendix D).

F.3 Composition directives

Composition experiments append a directive to the standard revision prompt. In the figures, A denotes the standard prompt and B adds the directive. The helper-composition directive is used in composition-prompt training and the directed evaluations in Figure 13. The task-specific helper directive is used in the composition-reward experiments; it excludes generic code-execution tools and requests answer checks and helper self-tests. Each comparison fixes the directive, with exceptions documented in Appendix C.4.

Helper-composition directive (abridged)

```
## Revision strategy directive (this round)
- **Compose the harness from helper tools.** Write one or a few helper
    functions that do the heavy computation for this task family, and expose
    them to the solver as function tools; the solver reads the question,
    extracts the facts, calls the tools, and reports the answer.
- **Keep every helper generic over the family.** A helper takes structured
    arguments (a grid, a number, a list of moves, an expression) and returns
    a computed result; it never parses prose and never depends on the exact
    wording of the question.
- **Run a real tool loop.** Pass `tools=[...]` and `return_message=True` to `
    llm_acall`, execute every tool call the solver returns, feed each result
    back as a tool message, stop after a few turns, and extract the final <
    answer> from the solver's last reply.
[...]
```

Task-specific helper directive (abridged)

```
## Revision strategy directive (this round)
[...]
- **No generic code-execution tool.** Do not expose `run_python`, `exec` or `
    eval` of solver-written code: that hands the solver an interpreter and
    contributes no domain logic. A real tool has a task-specific signature,
    for example `shortest_path(grid, start, goal)`, `count_primes(lo, hi)`, `
    is_valid_sudoku(grid)`, `simulate_moves(state, moves)`.
[...]
- **Check the answer before submitting it.** Verify the solver's answer
    string against the task's own constraints with a helper (simulate the
    moves to the goal, evaluate the expression against its target, confirm
    the count); if the check fails, tell the solver what failed and let it
    revise once.
[...]
- **Self-test each helper** on one small hand-checked instance before relying
     on it; a helper that throws on every question leaves the solver without
    tools.
[...]
```