METACURATE: EVOLVING CURATION FILTERS FOR HIGH-QUALITY AGENTIC CODING DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

Supervised fine-tuning (SFT) on agent trajectories is an effective way to improve coding agents, but selecting useful training examples requires more than identifying successful runs. A resolved patch may follow inefficient reasoning or unnecessary tool use, while a failed run may still contain useful intermediate steps. Training pipelines therefore rely on rules and qualitative assessments to filter trajectories, yet designing these filters typically requires substantial human expertise, and validating improvements through repeated training and evaluation is expensive. We introduce metacurate, a training-free framework that evolves executable trajectory curation filters using pairwise LLM judgments. Disagreements between the filter's rankings and the judge's preferences provide targeted feedback for jointly revising its rules, rubric, and aggregation policy. The resulting filter scores trajectories independently, so pairwise comparisons are required only during search. Starting from a blank filter, METACURATE autonomously discovers curation strategies that outperform a strong hand-authored filter refined through human expertise. We validate downstream utility through controlled SFT experiments on our uncurated trajectory pool and the Nemotron v3 SWE dataset, with random selection and hand-authored filtering as baselines. In our experiments, METACURATE yields the best performance among the compared methods on SWE-bench Verified, comparing to hand-authored filter and the filter crafted with $3\times$ expensive filter training-loop.

1 INTRODUCTION

Large language models (LLMs) have made rapid progress in coding, surpassing top human experts on some competitive programming benchmarks (Li et al., 2022). While early work focused mainly on generating code from a task description in a single response (Chen et al., 2021), coding agents now solve tasks through repeated interaction with an environment (Shen et al., 2026; Ludwig et al., 2026). Approaches such as ReAct (Yao et al., 2023) interleave reasoning with action; providing a framework for the repository exploration, code editing, and test-driven revision involved in agentic software engineering (Jimenez et al., 2024; Yang et al., 2024). Recent work has extended this iterative problem-solving capability to the model-training process itself, so that agents construct training data, launch training runs, and use evaluation results to design the next experiment, taking on research tasks previously carried out by humans (Rank et al., 2026; Yu et al., 2026; Yano et al., 2025; Karpathy, 2026).

Improving a model for agentic coding involves two separate steps: obtaining trajectories and choosing which of them to learn from (Zeng et al., 2026b; Liu et al., 2026). Supervised fine-tuning (SFT) provides a practical route to transferring capable agent behavior without repeated on-policy environment interactions during optimization (Shen et al., 2026; Pan et al., 2025; Peng et al., 2026). The most common selection rule keeps only the trajectories whose final patch passes the tests (i.e., rejection sampling; see Fig. 1 (a)) (Yang et al., 2025; Blakeman et al., 2026). Yet a trajectory that resolves a repository-level issue may still contain redundant exploration, malformed tool calls, or unproductive edit-and-test loops (Liu et al., 2025; Zhao et al., 2026), and a failed one may still contain a sound diagnosis or a useful test (Lv et al., 2026; Slinko et al., 2026). Unless such steps are removed or masked from the loss (Zeng et al., 2026a; Lab et al., 2026), SFT supervises them alongside the steps that led to success (Ma et al., 2026). Deciding what to keep as training data

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.

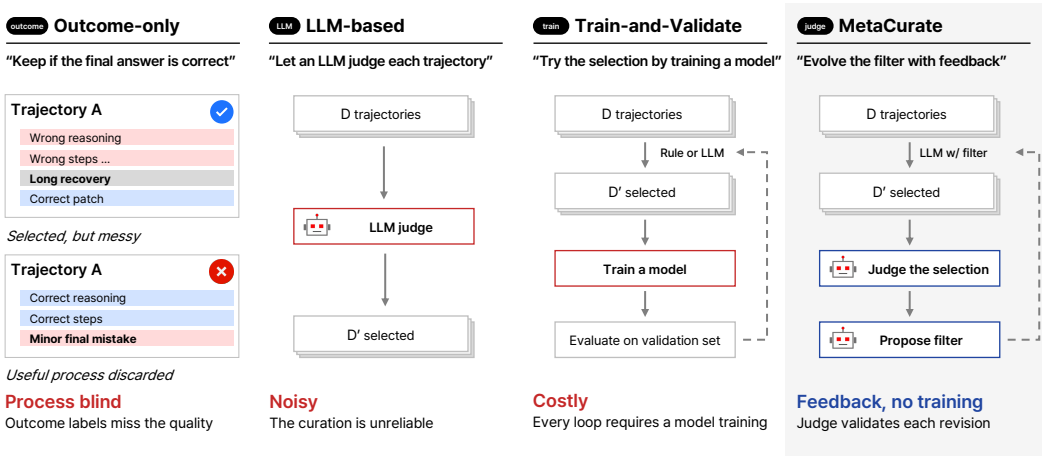

Figure 1: Four ways to select agent trajectories for SFT. (a) Outcome-only filtering keeps whatever passes the tests. (b) LLM-based filtering is noisy and unreliable. (c) Train-and-validate checks each selection with a training run, which is too expensive to iterate. (d) METACURATE validates each revision of the rules, rubric, and aggregation policy against pairwise judgments from an LLM judge, without training a model.

requires judging the quality of the problem-solving process together with its outcome (Han, 2026; Raoof et al., 2026).

In practice, therefore, agentic SFT data is selected by several kinds of judgment combined (Blakeman et al., 2026; Team et al., 2026; Lab et al., 2026). Deterministic rules check for recognizable failure patterns, semantic rubric criteria have LLM rate qualitative properties of the behavior, and an aggregation policy turns these outputs, together with the execution label, into a selection decision. This curation filter is usually written once and never validated on the pool it is applied to, and the scores an LLM assigns are noisy (see Fig. 1 (b)) (Pang et al., 2025; Kwok et al., 2026).

More recently, the curation strategy itself has become a target of automatic improvement (Wang et al., 2026a; Khan et al., 2025; Huang et al., 2026b; Yang et al., 2026a). In most of this work, the loop is closed by training: a model is trained on the data the strategy selects, and its failures or scores drive the next revision (see Fig. 1 (c)) (Wang et al., 2026a; Meng et al., 2026). This is the right signal, but every candidate then costs a training run (Lim et al., 2026; Kang et al., 2026), so we ask a complementary question: without placing model training inside the search loop, can a filter be improved iteratively from diagnostics and comparative feedback obtained on real trajectories?

We propose METACURATE, a training-free framework that iteratively improve an SFT data-curation filter (Fig. 1(d)). Rather than training a target model on the data selected by every candidate filter, METACURATE evaluates whether the candidate ranks trajectory pairs consistently with LLM judge's preference for their SFT usefulness. This provides a substantially cheaper feedback signal: each comparison requires only a judge call, allowing an iteration to evaluate many pairwise decisions instead of spending a training run on each candidate (Section 3.2). Pairwise judgments are also more directly actionable for revision, because each disagreement identifies two trajectories whose ordering is wrong and comes with the judge's rationale, whereas a downstream training score provides only a single aggregate outcome that must be attributed back to individual rules, rubric criteria, or their aggregation.

Using this feedback, METACURATE runs an agentic search over the filter itself. At each iteration, a proposer (i.e., LLM coding agent with access to the run directory) reads the current filter, its misranked trajectory pairs and judge rationales, and diagnostics of criterion health, and proposes revised candidates. A solver executes each candidate on sampled trajectories, while pairwise judgments accumulated during exploration provide feedback for subsequent revisions. To separate this search feedback from candidate selection, a revision is adopted only when its ranking improves agreement with judge preferences on a fixed comparison set hidden from the proposer by more than expected scoring noise. All intermediate filters, scores, judgments, and rationales are retained in the run directory and remain available to later iterations. Crucially, no target model is trained during this search:

pairwise comparisons supervise only the optimization of the filter. Once search is complete, the final filter independently scores every trajectory in the full pool, requiring neither a comparison partner nor an LLM pairwise judgment at deployment time.

We evaluate whether the improvements in concordance achieved by this search translate into better training-data selection. To this end, we roll out and release a corpus of 4,286 coding-agent trajectories produced by *DeepSeek V4 Pro* (Xu et al., 2026) on 1,415 real GitHub issues (Badertdinov et al., 2025; 2026; Pan et al., 2025), with up to three attempts per issue and an execution label for each trajectory. On this corpus and the 17,220-trajectory Nemotron v3 SWE-SFT pool (Blakeman et al., 2026), we conduct controlled SFT experiments with random selection as a control. On both pools, data selected by the evolved filter yields the highest observed mean SWE-bench Verified scores among the compared methods: 64.0 % and 56.5 %, compared with the competitive baselines filter. These results support improving the scoring instruction itself as a way to select effective training data without placing model training inside the search loop.

2 Related Work

Coding Agent. A coding agent resolves repository issues by inspecting code, running shell commands, editing files, and revising its changes based on test results (Yang et al., 2024). In a typical SWE-bench setup, the agent receives an issue description and works in a Docker container containing the repository and its dependencies; the final patch is evaluated using held-out repository tests (Jimenez et al., 2024). SFT on trajectories generated by stronger models in synthetic environments has helped open coding agents approach frontier performance (Pan et al., 2025; Yang et al., 2025). However, these datasets often include multiple attempts at the same issue, and even successful trajectories can contain unnecessary steps. Passing tests alone does not establish whether a trajectory is a useful training example (Yang et al., 2026b; Huang et al., 2026a). In practice, training pipelines use filters to select high-quality trajectories for SFT.

Data Curation. Existing curation methods use several signals to judge whether a trajectory is suitable for training. These include test outcomes (Yuan et al., 2023; Pan et al., 2025), soft verification against reference patches (Shen et al., 2026), and handwritten transcript heuristics (Blakeman et al., 2026; Team et al., 2025). Other methods use LLM judgments (Chen et al., 2024). These signals have limitations. LLM scores can contain model-dependent errors (Pang et al., 2025), rules can be redundant (Li et al., 2026), and heuristics can bias the selected data (Golubev et al., 2025). At scale, many selection methods offer little improvement over random sampling (Xia et al., 2025). Recent work therefore moves beyond fixed criteria and optimizes the curation procedure itself. It evolves curation code using the performance of models trained on the selected data as feedback (Wang et al., 2026a). MetaCurate instead jointly evolves a filter's rules, rubric, and aggregation policy. It evaluates candidate filters through agreement with a judge on held-out trajectory pairs, without training a model for each candidate.

3 Curation Signals and Scalable Feedback

3.1 Pair-wise Comparison Provides a Relative Reference

Comparative feedback has been widely used in open-ended reinforcement learning to address the weak discrimination and instability of absolute scores (Shao et al., 2024; Rafailov et al., 2023; Gunjal et al., 2026; Huang et al., 2025). Comparing two trajectories directly asks which is the more useful SFT example, with each providing a reference for assessing the other. This avoids requiring independently assigned quality scores to share a calibrated numerical scale (Viswanathan et al., 2025). Applying comparisons throughout a large pool can nevertheless be expensive: exhaustive comparison requires $\binom{N}{2}$ judgments for N trajectories. Sparse comparisons and tournaments reduce this cost, but repeated comparison-based ranking still incurs evaluation expense as the pool grows (Zhang et al., 2026; Yang et al., 2026c).

3.2 Point-wise Selection Scales, but Requires Validation

Evaluating trajectories individually makes pool-wide selection practical: for a fixed scoring procedure, the number of trajectory evaluations grows linearly with pool size (Chen et al., 2024; Liu et al., 2024). Outcome labels and LLM scores offer two ways to assess individual trajectories, while training-based feedback can validate and refine the resulting selection procedure (see Fig. 1).

Outcome Labels Leave Training Value Unresolved. Final task success alone does not establish the training value of the full trajectory (Yang et al., 2026b; Huang et al., 2026a). A passing run can contain redundant exploration or malformed tool calls (Liu et al., 2025; Zhao et al., 2026), while a failed run can contain a sound diagnosis or a useful test (Lv et al., 2026; Slinko et al., 2026). In our pool, all attempts receive the same outcome (i.e., resolved or non-resolved) on 1,275 of 1,415 issues (i.e., 90.1%), so the label cannot distinguish attempts within those issues. Among audited pairs with different outcome labels, LLM judge of SFT suitability prefers the failed trajectory in 28.3% of comparisons.

LLM Scores Require Empirical Validation. LLMs can assess qualitative properties beyond task success (Gunjal et al., 2026; Chen et al., 2024), but their scores alone do not establish whether a filter selects useful training examples or produces a meaningful ranking (Pang et al., 2025). Defining an appropriate evaluation standard (i.e., prompt) often requires people to inspect trajectories and formulate criteria based on experience (Blakeman et al., 2026; Team et al., 2025). Without explicit criteria or reference examples to anchor the scoring scale, the same trajectory may receive different scores across repeated evaluations, and different judges may disagree on which examples are preferable (Kwok et al., 2026; Zheng et al., 2023). Establishing reliable evaluation criteria therefore requires an iterative process of validation and refinement on real trajectories (Li et al., 2026; Wang et al., 2026b)

Training Feedback is Expensive and Coarse. Training on selected data evaluates its downstream effect, but repeated validation is expensive. Methods that optimize training-data selection through training–evaluation loops use downstream performance, sometimes supplemented by failure cases, to revise their curation strategies (Wang et al., 2026a; Khan et al., 2025; Meng et al., 2026). For example, evaluating three candidate strategies over three outer-loop iterations requires 110.2 B200-hours, approximately $659 of compute,[1] limiting the number of candidates we could evaluate (see Fig. 2) (Wang et al., 2026a). Performance scores and failure cases help diagnose the trained model, but do not directly attribute its behavior to a particular selection rule, rubric criterion, or aggregation policy. The search therefore needs feedback that is cheaper to obtain and more directly connected to the filter's selection decisions.

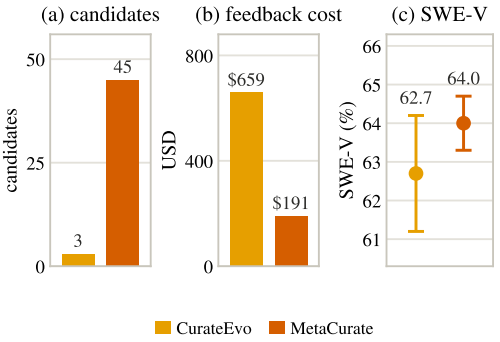

Figure 2: Training feedback costs 3.5 times as much and evaluates only 3 candidates against 45.

Design Implications. Curation search therefore requires a selection procedure that scales to the full pool and feedback that can evaluate that procedure and suggest revisions without repeatedly training a model(Section 4).

4 MetaCurate: Evolving the Curation Filter

Improving a filter without repeated model training requires a feedback signal that can both evaluate its selections and guide its revision. MetaCurate uses pairwise judge preferences as this

[1]The compute cost is estimated using RunPod (https://console.runpod.io/)'s B200 GPU pricing.

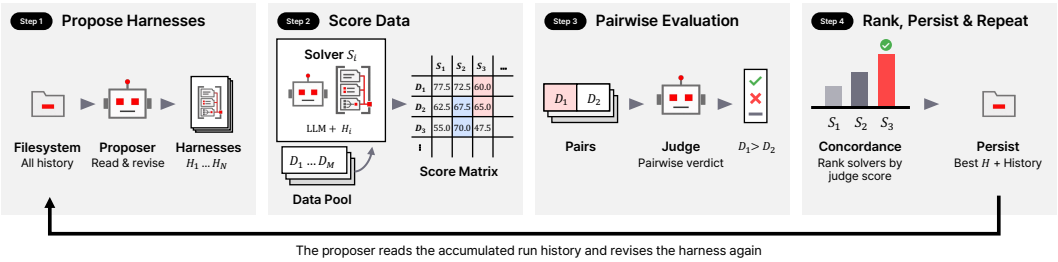

Figure 3: One iteration of METACURATE. The proposer edits the current filter and submits candidate filters (i.e., rules, rubric, and policy). The solver applies each candidate to the pool and gives every trajectory a score, which orders the pool. The candidate whose ordering agrees most with these verdicts is kept (see Section 4.3 for the promotion rule), and all scores, verdicts, and rationales are written to the file system for the next proposer.

supervision, transferring comparison feedback into a reusable pointwise filter. A filter is an executable scorer comprising three files: `rule.py`, `rubric.md`, and `policy.py`, which specify deterministic rules, LLM-evaluated criteria, and score aggregation. The loop evaluates candidates through comparison, edits them using actionable failure feedback, screens uncertain improvements, and applies the final filter to the full data pool (Fig. 3; Algorithm 1).

4.1 PAIRWISE JUDGMENTS FOR CURATION FILTER EVALUATION

We evaluate each candidate filter by comparing its trajectory rankings with the judge's pairwise preferences for SFT usefulness, rather than by training a model on the data it selects (Wang et al., 2026a; Khan et al., 2025). This agreement serves as a proxy for curation quality during search, while downstream SFT evaluates the utility of the selected data. Section 3 motivates using comparative judgments.

Each filter F scores a trajectory x individually, producing $s_F(x)$. For a judged pair $(x \succ y)$, let $\kappa_F(x, y)$ equal 1 if $s_F(x) > s_F(y)$, $\frac{1}{2}$ for a tie, and 0 otherwise. Its concordance with a preference set P is

$$C_F(P) = \frac{1}{|P|} \sum_{(x \succ y) \in P} \kappa_F(x, y). \tag{1}$$

Our objective is to maximize $C_F(P_{\text{sel}})$ subject to constraints on filter size, criterion redundancy, label use, and permitted edits. The selection set remains fixed throughout search, and both its trajectories and the associated judge preferences remain hidden from the proposer. This selection set is a development split held out from the same trajectory pool the filter curates, disjoint from the exploration rows, and it is not the downstream benchmark; no benchmark instance is read inside the loop. The proposer is also not shown $C_F(P_{\text{sel}})$ for any candidate, so a candidate's standing on these pairs cannot itself be used as feedback for the next edit. Constraints either reject a candidate before scoring or require stronger evidence for promotion.

Pairwise supervision is concentrated on sampled trajectories, while its reusable outcome is a pointwise scoring procedure. Once fixed, the filter can score the full pool without further pairwise judgments. Thus, comparisons supervise the construction of a filter whose application no longer requires a comparison partner for each trajectory.

4.2 CURATION FILTER EVOLUTION LOOP

The loop separates exploration feedback, which guides editing, from selection preferences, which evaluate candidates (Algorithm 1). At each iteration, a proposer reads the current filter, accumulated feedback from visible exploration pairs, and available criterion diagnostics. It proposes a few candidates, each changing one mechanism. Within its turn, and before submitting any candidate, the proposer repeatedly drafts an edit, executes it on a few exploration trajectories, reads the result, and redrafts, so that an edit is measured rather than guessed. This inner revision is offline and uses no judge preferences; it is distinct from the single revision round the precheck grants a submitted candidate. The solver scores the sampled trajectories with the valid candidates. The judge then

Algorithm 1 METACURATE outer loop over filters

Input: exploration set D_{exp}, hidden selection pairs P_{sel}, proposer P, iterations N
Initialize: current filter $F \leftarrow F_0$
Initialize: run directory $\mathcal{D} \leftarrow \emptyset$
for $t = 1 \dots N$ **do**
  Proposer P queries $\mathcal{D}$ ▷ prior filters, scores, feedback
  repeat
    Proposer edits F and probes the draft on D_{exp}
    Store probe results in $\mathcal{D}$
  until probe budget is spent
  Proposer submits k candidate filters $\{F_1, \dots, F_k\}$
  Precheck candidates and revise rejected candidates once
  for $F_i \in \{F_1, \dots, F_k\}$ **do**
    if F_i passes validation **then**
      $E_i \leftarrow \text{EVALUATE}(F_i, D_{\text{exp}}, P_{\text{sel}})$
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(F_i, E_i)\}$
  $F \leftarrow \text{PROMOTE}(F, \mathcal{D})$ ▷ best hidden-pair concordance, gated
return SFT subset selected by F

adjudicates exploration pairs on which candidate orderings disagree; pairs all candidates order alike are skipped. These verdicts and their rationales accumulate as feedback for subsequent edits, but do not determine promotion. Candidates are instead compared on the fixed hidden selection pairs under the promotion rule below. Exploration evidence, diagnostics, and promotion records persist across iterations, making previous outcomes available for subsequent proposals. When search ends, the final filter scores the full pool to select the SFT subset.

4.3 IMPLEMENTATION DETAILS

Actionable failure feedback. The proposer receives the number and examples of exploration pairs the filter misorders, together with the judge's comparative rationales. These identify concrete failures that revisions to rules, rubric criteria, or aggregation can address. Criterion score distributions flag criteria with little variation (Wang et al., 2026b; Guan et al., 2026), while pairwise Spearman correlations flag potentially redundant rubric or rule criteria (Li et al., 2026). These empirical indicators of criterion diversity suggest which criteria to repair, remove, differentiate, or merge.

Probing during proposal. Proposer runs a draft `rules.py` over exploration rows and sees each rule's firing rate; a rule that fires on none or nearly all of them orders nothing, and without this the loop spends an iteration before that shows. A rubric criterion instead needs the solver, so the proposer scores a draft criterion on a few exploration rows under the run's own solver model and prompt, under a budget that affords tens of such calls per turn. The proposer therefore iterates on a draft as many times as the evidence warrants. Both tools refuse any trajectory outside D_{exp}, leaving the selection set unreachable.

Order-consistent judgments. Because LLM judges can exhibit position bias (Wang et al., 2024; Zheng et al., 2023; Shi et al., 2025), every judged pair is presented in both AB and BA order. We retain a preference only when both presentations identify the same winner.

Noise-aware promotion. LLM scoring variability makes a small concordance gain insufficient evidence of improvement (Ouyang et al., 2026; Pang et al., 2025). Promotion therefore uses a one-sided test of the selection pairs on which the candidate and current filter differ in correctness, rather than simply taking the highest observed score. Candidates introducing new criterion-health defects face a stricter threshold. If none of the conditions are satisfied, the current filter is retained.

Pointwise scoring. Rubric criteria use anchors from -2 to $+2$ and are evaluated in separate LLM calls, each containing the preamble and one criterion, to reduce conflation of assessment dimensions (Kwok et al., 2026; Viswanathan et al., 2025). The aggregation policy combines rubric scores and deterministic features into a score in $[0, 100]$.

Table 1: Downstream performance of *Qwen 3.5 9B* across curation methods on two pools, with a default training budget of 1,000 trajectories. Scores are SWE-bench Verified resolved rates (%). The second column lists hand-authored guidance or oracle information required for curation.

Method	Hand-authored or Oracle	DSv4Pro	Nemotron v3
Base model	—	53.9±2.0	53.9±2.0
Full pool (volume anchor)	N/A	63.1±1.2	54.0±1.6
Random	N/A	61.7±1.6	52.9±1.3
Resolved (Yuan et al., 2023)	Execution labels	61.9±0.7	54.4±0.2
Gold-patch recall (Shen et al., 2026)	Reference patches	61.7±2.4	53.9±0.5
Rule filter (Blakeman et al., 2026)	7 signals, 6 thresholds	63.1±1.4	54.4±0.5
LLM filter (Chen et al., 2024)	1 prompt	62.9±1.2	54.8±2.2
CurateEvo (Wang et al., 2026a)	N/A	62.7±1.5	53.6±0.6
Ours, handwritten seed	Handwritten 6 criteria	63.3±0.6	54.7±0.4
Ours	**N/A**	**64.0±0.7**	**56.5±1.7**

5 EXPERIMENTS

5.1 SETUP

The main comparison evaluates the downstream utility of METACURATE-selected data against alternative curation methods. Unless otherwise specified, Ours refers to the blank-seed variant of METACURATE. DSv4Pro corpus contains 4,286 trajectories generated for this study by *DeepSeek V4 Pro* using mini-swe-agent. The Nemotron v3 pool contains 17,220 trajectories drawn exclusively from the mini-swe-agent subset of the Nemotron v3 SWE-SFT release (Blakeman et al., 2026). We train *Qwen 3.5 9B* for all methods, with a shared fine-tuning recipe within each pool. The default training-set budget is 1,000 trajectories; exceptions for the Full pool baseline and the DSv4Pro Gold-patch recall arm are described in Appendix A.

METACURATE uses *GPT-5.6 Sol Pro (Sol Pro)* as proposer and *GPT-5.6 Luna (Luna)* as scorer and pairwise judge. The LLM-based baselines also use *Luna* for trajectory scoring. Downstream evaluation uses mini-swe-agent on all 500 SWE-bench Verified instances. Main-comparison scores report the mean and standard deviation over three evaluations of each trained checkpoint. Baseline definitions and protocol details are in Appendix A, and training and evaluation settings are in Table 6.

5.2 MAIN COMPARISON

Training-free curation achieves the highest mean downstream scores. Our adaptation of CurateEvo (Wang et al., 2026a) trains models inside its curation loop, while METACURATE evaluates candidate filters using pairwise judge preferences. At a budget of 1,000 trajectories, *Qwen 3.5 9B* trained on data selected by the blank-seed filter reaches 64.0% on DSv4Pro and 56.5% on Nemotron v3, compared with 62.7% and 53.6% for CurateEvo, respectively (Table 1). METACURATE thus obtains the highest observed means among the compared methods on both pools without model training during filter search. Starting from a single generic criterion, METACURATE revises the rules, rubric, and aggregation policy from feedback on real trajectories, without execution labels, reference patches, or a task-specific handwritten initial filter. Random provides a control for subset selection: its mean scores at the same 1,000-trajectory budget are 2.3 and 3.6 points below METACURATE on DSv4Pro and Nemotron v3, respectively. On DSv4Pro, the selected subset also achieves performance comparable to training on all 4,286 trajectories (63.1%).

5.3 MODEL SCALE

To assess whether METACURATE's benefits extend beyond the 9B model, *Qwen 3.5 27B* is trained on the same Random, Resolved, Rule filter, and Ours subsets, without repeating filter search. Ours achieves a mean resolved rate of 74.3% on SWE-bench Verified, compared with 73.3% for Resolved, 73.1% for Rule filter, and 72.7% for Random (Table 2). The gain over the compared baselines extends to 27B using the same selected data, without repeating filter search.

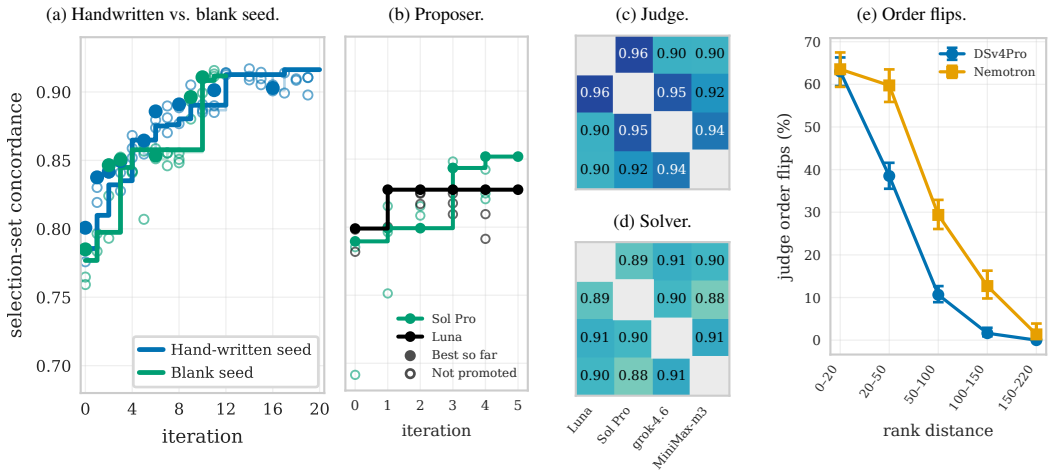

Figure 4: **Curation-loop ablations.** (a) A blank seed achieves concordance comparable to a hand-written filter, without task-specific handwritten initialization. (b) *Sol Pro* achieves higher concordance than *Luna* as proposer with the same solver and judge. (c, d) Judge preferences and solver rankings largely agree across models. Both matrices follow the model order shown in (d). (e) Presentation-order flips are more frequent for closely ranked trajectories.

Table 2: Transfer to *Qwen 3.5 27B* using the same training subsets as for 9B.

Method	Resolved (%)
Base	72.3±0.6
Random	72.7±1.2
Resolved	73.3±0.6
Rule filter	73.1±1.0
Ours	**74.3±0.6**

Table 3: Iteration series on DSv4Pro. SWE-bench Verified scores are mean ± SD over three evaluations.

Iteration	3	6	9	Final
Concordance	0.845	0.851	0.856	**0.912**
Resolved (%)	62.4±1.2	62.0±0.8	63.3±0.5	**64.0±0.7**

5.4 Ablation Studies

Initial filter ablation. To test whether MetaCurate depends on a handwritten initial filter, Fig. 4(a) compares the deployed filter of Section 3 with a blank filter initialized from a single generic criterion. Both starting points improve during search, reaching best observed selection concordances of 0.916 for the handwritten seed over twenty iterations and 0.912 for the blank seed over twelve. The two are level on this measure, and in the main comparison the blank seed yields higher mean downstream scores than the handwritten seed on both DSv4Pro (64.0% vs. 63.3%) and Nemotron v3 (56.5% vs. 54.7%; Table 1). These results further support obtaining useful training data without a task-specific handwritten initial filter.

Iteration series. The iteration series tests whether higher selection concordance is accompanied by better downstream performance. *Qwen 3.5 9B* is trained separately on the 1,000 DSv4Pro trajectories selected at iterations 3, 6, 9, and the end of a single blank-seed run (Table 3). As selection concordance rises from 0.845 to 0.912, the mean resolved rate increases overall from 62.4% to 64.0%. The final filter achieves the highest values on both measures. This trend provides preliminary support for selection concordance as a search signal for downstream data utility. SWE-bench Verified is reserved for downstream evaluation: no benchmark instance enters filter search, and no benchmark score is available to either the proposer or the promotion rule. The downstream gains across search iterations (Table 3) further suggest that improvements in selection-set concordance translate into better performance, rather than merely reflecting overfitting to the selection set.

Loop component ablation. *Luna* and *Sol Pro* each optimize a filter from the same blank seed, with *Luna* fixed as both solver and judge. After five iterations, *Sol Pro* achieves higher selection concordance with the fixed judge's preferences than *Luna* (Fig. 4(b)). This supports using *Sol Pro* as proposer to better optimize the filter for the same solver. Judge agreement is measured on 100 trajec-

tory pairs, each evaluated in both presentation orders by *Luna*, *Sol Pro*, *Grok-4.6*, and *MiniMax-m3*. Their pairwise agreement ranges from 0.90 to 0.96 on pairs for which both judges select a consistent winner across presentation orders (Fig. 4(c)). This broad agreement motivates using *Luna*, whose calls cost approximately $0.004 each compared with $0.14 for *Sol Pro*. Using the same final filter optimized from the blank seed, the four solver models score 220 trajectories. Their resulting rankings agree on 0.88 to 0.91 of pairs with no tied scores from either model (Fig. 4(d)). These broadly similar rankings motivate using *Luna*, which costs approximately $0.004 per scoring call compared with $0.09 for *Sol Pro*.

These results support using a stronger proposer for filter optimization, while relying on a lower-cost model for pairwise judging and scoring with the optimized filter.

Order flips against rank distance. When two trajectories are close in rank, swapping their presentation order often changes the judge's preference. Preferences are more stable for trajectories farther apart (Fig. 4(e)). This pattern supports judging each pair in both orders and retaining only preferences that remain consistent (Section 4.1).

Table 4: 200-trajectory budget on DSv4Pro.

Method	Resolved (%)
Random	58.7±1.1
Resolved	59.8±1.4
Gold-patch recall	58.1±1.2
Rule filter	59.8±1.1
LLM filter	58.8±0.9
CurateEvo	58.5±0.6
Ours	**60.3±2.1**

Training-set budget. To test curation with less training data, we reuse the blank-seed filter from the main comparison to select the top 200 DSv4Pro trajectories. A *Qwen 3.5 9B* model trained on this subset reaches 60.3%, the highest observed mean, versus 58.7% for Random and 59.8% for the strongest baselines (Table 4).

6 Conclusion

We introduced MetaCurate, a training-free framework for improving trajectory curation filters through pairwise LLM feedback and criterion diagnostics. Rather than treating curation as a fixed set of human-designed rules or validating each revision through costly model training, MetaCurate iteratively revises deterministic rules, rubric criteria, and score aggregation, producing an executable filter that can score trajectories independently at scale. Across two coding-trajectory datasets, the resulting curation filters select subsets that yield stronger downstream models than the compared baselines, showing that pairwise judgments can provide a useful signal for improving pointwise data selection. More broadly, our results suggest that curation itself can be systematically inspected, revised, and validated before training, providing a practical and interpretable approach to selecting useful SFT trajectories.

Limitations and Future work Our experiments focus primarily on whether the filters evolved by MetaCurate improve downstream data selection, rather than on interpreting the curation principles that emerge during search. An important direction for future work is therefore to analyze how the executable filter changes over time, including which rules and rubric criteria are introduced, removed, or repeatedly strengthened. Relating these changes to downstream training outcomes could provide a more quantitative account of which trajectory properties make an example useful for SFT. Controlled analyses of individual rules or criteria could further reveal whether the behaviors preferred by the evolved filter are subsequently reflected in the trained agent. Ultimately, such analyses may turn MetaCurate from a procedure for selecting training data into a tool for extracting empirical principles that can inform how future agent trajectories are collected, generated, and curated.

AI use statement

LLMs are part of the method studied in this paper, and that use is reported in the body. *GPT-5.6 Sol Pro*, run as a coding agent through the Codex CLI, proposes filter revisions, and *GPT-5.6 Luna* scores each criterion and judges trajectory pairs. The trajectories that make up the pools were generated by *DeepSeek V4 Pro* (Section 3) and by the models behind the Nemotron v3 SWE-SFT release (Section 5). Beyond the method, we used LLM coding assistants to write and debug the

experiment harness, the analysis scripts, and the figure-generation code, and to edit the prose of this paper for clarity and length. The research questions, the diagnosis of Section 3, the design of the loop, the experimental design, and the interpretation of results are the authors' own.

Ethics statement

This work involves no human subjects and collects no personal data. The trajectories we curate are solutions to public GitHub issues drawn from SWE-rebench and SWE-Gym, and from the public Nemotron v3 SWE-SFT release, and we use them under the terms of those releases. We did not use any GitHub pull requests that we collected ourselves. The artifacts we release are row identifiers, per-row scores, and the judge's pairwise verdicts, which reference rows of the public pools rather than copy them. The method optimizes a filter against an LLM judge, so the filter inherits whatever the judge prefers, including preferences that do not track downstream quality. The paper reports where the judge's signal and the downstream result agree and where they do not, and the loop is designed to reject changes that the judge cannot separate from noise. A curated set can still carry the judge's blind spots into the solver, and users of such filters should evaluate the trained model on the tasks they care about rather than trust the filter's own score. Benchmark integrity is the risk we take most seriously. The loop's selection set is disjoint from SWE-bench Verified, and no benchmark result is read inside the loop. Both pools pass a gate that rejects any trajectory whose instance id or repository and commit appear in SWE-bench Verified, and on the DSv4Pro sources the gate found no overlap.

References

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0788. URL https://openreview.net/forum?id=nMpJoVmRy1.

Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, and Alexander Golubev. Swe-rebench v2: Language-agnostic swe task collection at scale. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=UCAda9kS57.

Aaron Blakeman, Aaron Thomas, Aastha Jhunjhunwala, Abhibha Gupta, Abhinav Khattar, Adam Rajfer, Adi Renduchintala, Adil Asif, Aditya Vavre, Adriana Flores Miranda, et al. Nemotron 3 ultra: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2606.15007*, 2026.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, et al. Alpagasus: Training a better alpaca with fewer data. In *International Conference on Learning Representations*, volume 2024, pp. 34767–34797, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Alexander Golubev, Maria Trofimova, Sergei Polezhaev, Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Sergey Abramov, Andrei Andriushchenko, Filipp Fisin, et al. Training long-context, multi-turn software engineering agents with reinforcement learning. *arXiv preprint arXiv:2508.03501*, 2025.

Xin Guan, Xiaomeng Hu, Shen Huang, Zhenyi Wang, Bo Zhang, Zijian Li, Pengjun Xie, Bo Liu, and Jiuxin Cao. Evorubric: Self-evolving rubric-driven rl for open-ended generation. *arXiv preprint arXiv:2605.29847*, 2026.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *International Conference on Learning Representations*, volume 2026, pp. 127924–127945, 2026.

Yunze Han. A systematic evaluation of trajectory data curation for LoRA fine-tuning of code agents. In *Advanced Intelligent Computing Technology and Applications: 22nd International Conference on Intelligent Computing, ICIC 2026, Toronto, ON, Canada, July 22–26, 2026, Proceedings, Part XXVIII*, pp. 39–50, Berlin, Heidelberg, 2026. Springer-Verlag. ISBN 978-981-92-3491-2. doi: 10.1007/978-981-92-3492-9_4. URL https://doi.org/10.1007/978-981-92-3492-9_4.

Jiawei Huang, Qingping Yang, Renjie Zheng, and Jiaze Chen. Beyond verifiable rewards: Rubric-based grm for reinforced fine-tuning swe agents. *arXiv preprint arXiv:2604.16335*, 2026a.

Wei Huang, Anda Cheng, Yinggui Wang, Lei Wang, and Tao Wei. Llm-autodp: Automatic data processing via llm agents for model fine-tuning. *arXiv preprint arXiv:2601.20375*, 2026b.

Zenan Huang, Yihong Zhuang, Guoshan Lu, Zeyu Qin, Haokai Xu, Tianyu Zhao, Ru Peng, Jiaqi Hu, Zhanming Shen, Xiaomeng Hu, et al. Reinforcement learning with rubric anchors. *arXiv preprint arXiv:2508.12790*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Feiyang Kang, Hanze Li, Adam Nguyen, Mahavir Dabas, Jiaqi W Ma, Frederic Sala, Dawn Song, and Ruoxi Jia. Can generalist agents automate data curation? *arXiv preprint arXiv:2606.04261*, 2026.

Andrej Karpathy. AutoResearch: AI agents running research on single-GPU nanochat training automatically. GitHub repository, 2026. URL https://github.com/karpathy/autoresearch. Accessed 2026-09-22.

Zaid Khan, Elias Stengel-Eskin, Jaemin Cho, and Mohit Bansal. Dataenvgym: Data generation agents in teacher environments with student feedback. In *International Conference on Learning Representations*, volume 2025, pp. 53480–53507, 2025.

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Yixing Jiang, Chelsea Finn, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. Llm-as-a-verifier: A general-purpose verification framework. *arXiv preprint arXiv:2607.05391*, 2026.

Nanbeige Lab, Chen Yang, Chengrui Huang, Fufeng Lan, Hanhui Chen, Hao Zhou, Huatong Song, Jiaqi Cao, Jiaying Zhu, Jinlin Niu, et al. Nanbeige4. 2-3b: Unlocking agentic capabilities in a compact model. *arXiv preprint arXiv:2607.22083*, 2026.

Xiaomin Li, Mingye Gao, Zhiwei Zhang, Chang Yue, and Hong Hu. Selection of llm fine-tuning data based on orthogonal rules. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40 (37):31760–31768, 2026.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Joy Jia Yin Lim, Xin Huang, Hao Peng, Yaxi Lu, Xin Cong, Zhong Zhang, Maosong Sun, and Yankai Lin. What is missing from ai post-training ai: An empirical analysis. *arXiv preprint arXiv:2608.19072*, 2026.

Simiao Liu, Fang Liu, Liehao Li, Xin Tan, Yinghao Zhu, Xiaoli Lian, and Li Zhang. An empirical study on failures in automated issue solving. *arXiv preprint arXiv:2509.13941*, 2025.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. In *International Conference on Learning Representations*, volume 2024, pp. 22353–22373, 2024.

Wei Liu, Siya Qi, Yali Du, and Yulan He. Position: Self-play only evolves when self-synthetic pipeline ensures learnable information gain. In *Forty-third International Conference on Machine Learning Position Paper Track*, 2026. URL https://openreview.net/forum?id=xymZU1Fd6Z.

Nikolai Ludwig, Wasi Uddin Ahmad, Somshubra Majumdar, and Boris Ginsburg. From swe-zero to swe-hero: Execution-free to execution-based fine-tuning for software engineering agents. *arXiv preprint arXiv:2604.01496*, 2026.

Dongyi Lv, Aichen Cai, Liang Huang, Ya Zhang, Qiuyu Ding, Canhui Wu, Zhi Wang, Yuesong Zhang, Jiaqi Wang, Nan Duan, et al. Failforge: Distilling procedural competence from persistent failures into code agents. *arXiv preprint arXiv:2608.08570*, 2026.

Murong Ma, Tianyu Chen, Yun Lin, Shuai Lu, Qinglin Zhu, Yeyun Gong, Zhiyong Huang, Peng Cheng, Yan Lu, and Jin Song Dong. From patches to trajectories: Privileged process supervision for software-engineering agents. *arXiv preprint arXiv:2605.21996*, 2026.

Fanqing Meng, Lingxiao Du, Qiguang Chen, Ziqi Zhao, Haocheng Lu, Mengkang Hu, and Michael Qizhe Shieh. Rsibench-data: Benchmarking data-centric research for recursive self-improvement. *arXiv preprint arXiv:2607.25886*, 2026.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=Cq1BNvHx74.

Jinlong Pang, Jiaheng Wei, Ankit Parag Shah, Zhaowei Zhu, Yaxuan Wang, Chen Qian, Yang Liu, Yujia Bao, and Wei Wei. Improving data efficiency via curating llm-driven rating systems. In *International Conference on Learning Representations*, volume 2025, pp. 100986–101016, 2025.

Baolin Peng, Wenlin Yao, Qianhui Wu, Hao Cheng, Xiao Yu, Rui Yang, Tao Ge, Alessandro Sordoni, Xingdi Yuan, Yelong Shen, et al. Orchard: An open-source agentic modeling framework. *arXiv preprint arXiv:2605.15040*, 2026.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, 2023.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. Posttrainbench: Can LLM agents automate LLM post-training? 2026. URL https://openreview.net/forum?id=UnjxMTe57e.

Negin Raoof, Richard Zhuang, Marianna Nezhurina, Etash Guha, Atula Tejaswi, Ryan Marten, Charlie F Ruan, Tyler Griggs, Alexander Glenn Shaw, Hritik Bansal, et al. Openthoughts-agent: Data recipes for agentic models. *arXiv preprint arXiv:2606.24855*, 2026.

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.

Ethan Shen, Daniel Tormoen, Saurabh Shah, Ali Farhadi, and Tim Dettmers. SERA: Soft-verified efficient repository agents. 2026. URL https://openreview.net/forum?id=GHKj1XcSlt.

Lin Shi, Chiyu Ma, Wenhua Liang, Xingjian Diao, Weicheng Ma, and Soroush Vosoughi. Judging the judges: A systematic study of position bias in llm-as-a-judge. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 292–314, 2025.

Igor Slinko, Ilia Zavidnyi, Egor Bogomolov, and Yaroslav Zharov. Step rejection fine-tuning: A practical distillation recipe. *arXiv preprint arXiv:2605.10674*, 2026.

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

NeoHorse Team, Guoliang Cao, Guohao Dai, Tianyu Guo, Kai Han, Hailin Hu, Zihan Jiang, Xiang Kuang, Boxun Li, Yulong Li, et al. Neohorse-1: Towards recursive self-improvement via agentic post-training with routing harness. *arXiv preprint arXiv:2609.08183*, 2026.

Vijay Viswanathan, Yanchao Sun, Xiang Kong, Meng Cao, Graham Neubig, and Sherry Wu. Checklists are better than reward models for aligning language models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 114728–114754, 2025. doi: 10.52202/085713-3824. URL https://openreview.net/forum?id=RPRqKhjrr6.

Dingzirui Wang, Xuanliang Zhang, Keyan Xu, Qingfu Zhu, and Wanxiang Che. Curateevo: Data-curation evolving for agentic post-training. *arXiv preprint arXiv:2607.06140*, 2026a.

Jianze Wang, Kunwang Zheng, Ying Liu, Yu Cao, Qilong Zhang, Jinlong Chen, Hua Yang, and Qianglong Chen. Serpo: Self-evolving rubric policy optimization for open-ended test-time reinforcement learning. *arXiv preprint arXiv:2607.26873*, 2026b.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, et al. Large language models are not fair evaluators. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 9440–9450, 2024.

Tingyu Xia, Bowen Yu, Kai Dang, An Yang, Yuan Wu, Yuan Tian, Yi Chang, and Junyang Lin. Rethinking data selection at scale: Random selection is almost all you need. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 2698–2711, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.146. URL https://aclanthology.org/2025.findings-emnlp.146/.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

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.

John Yang, Kilian Lieret, Carlos Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3239. URL https://openreview.net/forum?id=63iVrXc8cC.

Shidong Yang, Ziyu Ma, Tongwen Huang, Yiming Hu, Yong Wang, and Xiangxiang Chu. Coevolve: Training llm agents via agent-data mutual evolution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23015–23036, 2026a.

Yuming Yang, Mingyoung Lai, Wanxu Zhao, Xiaoran Fan, Zhiheng Xi, Mingqi Wu, Chiyue Huang, Jun Zhao, Haijun Lv, Jian Tong, et al. Which reasoning trajectories teach students to reason better? a simple metric of informative alignment. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 42123–42150, 2026b.

Zixuan Yang, Yiqun Chen, Wei Yang, Erhan Zhang, Zihan Shen, Xiaochi Wei, Yan Gao, Yi Wu, Yao Hu, and Jiaxin Mao. Tournament-grpo: Group-wise tournament rewards for reinforcement learning in open-ended long-form generation. *arXiv preprint arXiv:2605.26958*, 2026c.

Taro Yano, Yoichi Ishibashi, and Masafumi Oyamada. Lamdagent: An autonomous framework for post-training pipeline optimization via llm agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 30066–30083, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zhaojian Yu, Penghao Yin, Shuzheng Gao, Shilin He, Kai Cai, and Xiao-Ping Zhang. Autotrainess: Teaching language models to improve language models autonomously. *arXiv preprint arXiv:2606.31551*, 2026.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026a.

Xingshan Zeng, Zishan Xu, Boju Zhang, Yuzhou Wu, Lingzhi Wang, Jianghao Lin, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, et al. What makes good agentic data? an ace lens on data generation for llm agents. *arXiv preprint arXiv:2608.27260*, 2026b.

Qiang Zhang, Boli Chen, Fanrui Zhang, Ruixue Ding, Shihang Wang, Qiuchen Wang, Yinfeng Huang, Haonan Zhang, Rongxiang Zhu, Xin Li, Houquan Zhou, Pengjun Xie, Kaipeng Zhang, Jingren Zhou, and Jiawei Liu. ArenaRL: Scaling RL for open-ended agents via tournament-based relative ranking. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=1XUCkpEknR.

Xiangxin Zhao, Han Li, Shuaiting Li, Tianyi Zhao, Earl T Barr, Federica Sarro, and He Ye. Failure as a process: An anatomy of cli coding agent trajectories. *arXiv preprint arXiv:2607.09510*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023.

A Experimental Setup Details

Details of DSv4Pro *DeepSeek V4 Pro* (Xu et al., 2026) generated coding-agent trajectories for instances from SWE-rebench, SWE-rebench-V2, and SWE-Gym using mini-swe-agent (Yang et al., 2024) at temperature 0.7. The collection targeted three attempts per source instance and produced 4,286 stored trajectories (Table 5). Of the 1,415 instance IDs represented in the pool, 1,369 have three trajectories, 23 have two, one has a single trajectory, and 22 have six.

Table 5: Source composition of DSv4Pro.

Source dataset	Instance IDs	Trajectories
SWE-rebench (Badertdinov et al., 2025)	690	2,100
SWE-rebench-V2 (Badertdinov et al., 2026)	531	1,599
SWE-Gym (Pan et al., 2025)	196	587
Combined pool	1,415	4,286

SWE-Gym shares one instance each with SWE-rebench and SWE-rebench-V2. SWE-rebench and SWE-rebench-V2 share two more issues under differently capitalized IDs.

Each submitted patch was applied in the instance's container and evaluated with its fail-to-pass and pass-to-pass tests. A trajectory is labeled *resolved* when both test sets pass; 1,760 trajectories (41.1% of the pool) satisfy this condition.

Details of Nemotron v3 The source is NVIDIA's public Nemotron v3 SWE-SFT release (Blakeman et al., 2026). Only trajectories generated with mini-swe-agent are retained, giving a pool of 17,220 trajectories.

Training and evaluation details Table 6 lists the shared settings and model-specific training configurations. We report SWE-bench Verified on all 500 instances (Jimenez et al., 2024). Every agent uses mini-swe-agent (Yang et al., 2024), a bash-only variant of SWE-agent.

Baselines The standard training-set budget in Table 1 is 1,000 trajectories; the Full pool baseline and the DSv4Pro Gold-patch recall arm are exceptions described below.

- **Full pool.** The volume anchor trains on the entire source pool without filtering.
- **Random.** Random selects a random subset of 1,000 trajectories from the source pool.
- **Resolved.** Resolved keeps trajectories by execution label alone (Yuan et al., 2023; Pan et al., 2025; Yang et al., 2025).
- **Gold-patch recall.** Following SERA (Shen et al., 2026), this baseline filters trajectories by reference-patch line recall with a threshold of 0.5. On DSv4Pro, 629 training rows meet this threshold, so this arm trains on all 629 without lowering the threshold or padding the subset. On Nemotron v3, enough trajectories meet the threshold to select 1,000.
- **Rule filter.** The Rule filter applies the Nemotron v3 transcript heuristics (Blakeman et al., 2026).
- **LLM filter.** The LLM filter asks a judge for a quality verdict (Chen et al., 2024).
- **CurateEvo.** CurateEvo evolves a curation program against dev scores of models trained inside its own loop (Wang et al., 2026a). The run reported in Table 1 holds out 10% of the training set for validation, with *Luna* as its scorer. Its 1k arm is a seeded draw of 1,000 trajectories accepted by the evolved program.

Table 6: Training and evaluation settings. Shared settings are listed separately from each model's fine-tuning method and learning rate.

Training (Shared)	
Epochs	3
Schedule	cosine to 1×10^{-6}, warmup ratio 0.1
Optimizer	AdamW, $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, weight decay 0.01
Batch	32
Sequence length	128,000 tokens, no packing, loss on assistant turns only
Gradient clipping	1.0
Precision	bf16
Training (per model)	
9B	full fine-tuning, learning rate 1×10^{-5}
27B	LoRA $r = 64$, $\alpha = 128$, dropout 0, learning rate 1×10^{-4} attention and MLP projections, FSDP full shard
Evaluation	
Benchmark	SWE-bench Verified (500 instances)
Sampling	temperature 1.0, top-p 0.95, top-k 20
Output cap	16,384 tokens per turn
Step limit	250 agent steps per instance
Wall clock	6 hours per instance
Serving	vLLM, max model length 262,144
Runs	three replicates per cell, averaged; exceptions are marked in place

B Filter Evolution

Blank and handwritten seeds show different search patterns (Figs. 5 and 6). Darker red and blue indicate larger positive and negative score differences from the current filter. The blank-seed run produces more pronounced departures in both directions, whereas the handwritten-seed run has mostly pale cells and fewer large score differences. This pattern is consistent with greater inertia around an established handwritten filter, while a blank seed leaves more room for substantial changes in scoring behavior.

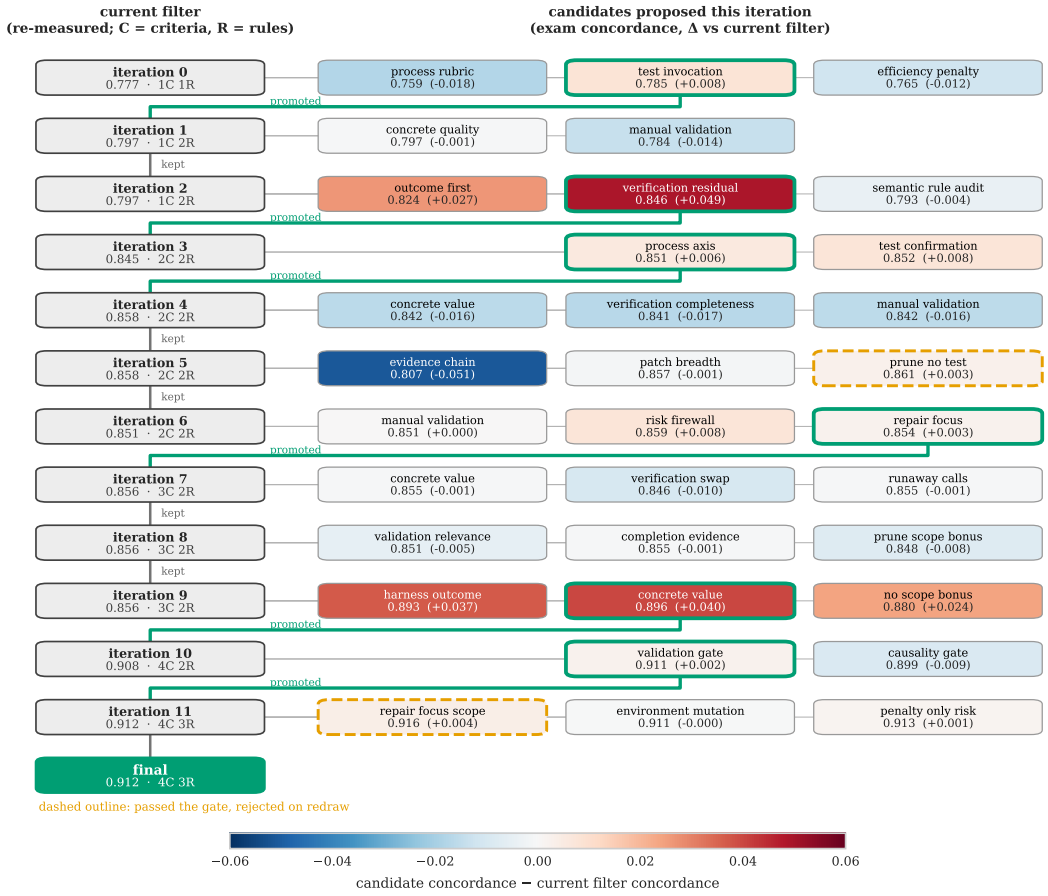

Figure 5: Search history from a blank seed over twelve iterations. Each row shows the current filter on the left and proposed candidates on the right; C and R denote the numbers of criteria and rules. Empty slots indicate candidates that did not pass precheck (Algorithm 1). Red and blue indicate positive and negative concordance differences from the current filter, with darker colors indicating larger differences. Green outlines mark promoted candidates. Orange dashed outlines mark eligible candidates in iterations where the provisionally selected candidate failed re-evaluation, leaving the current filter unchanged.

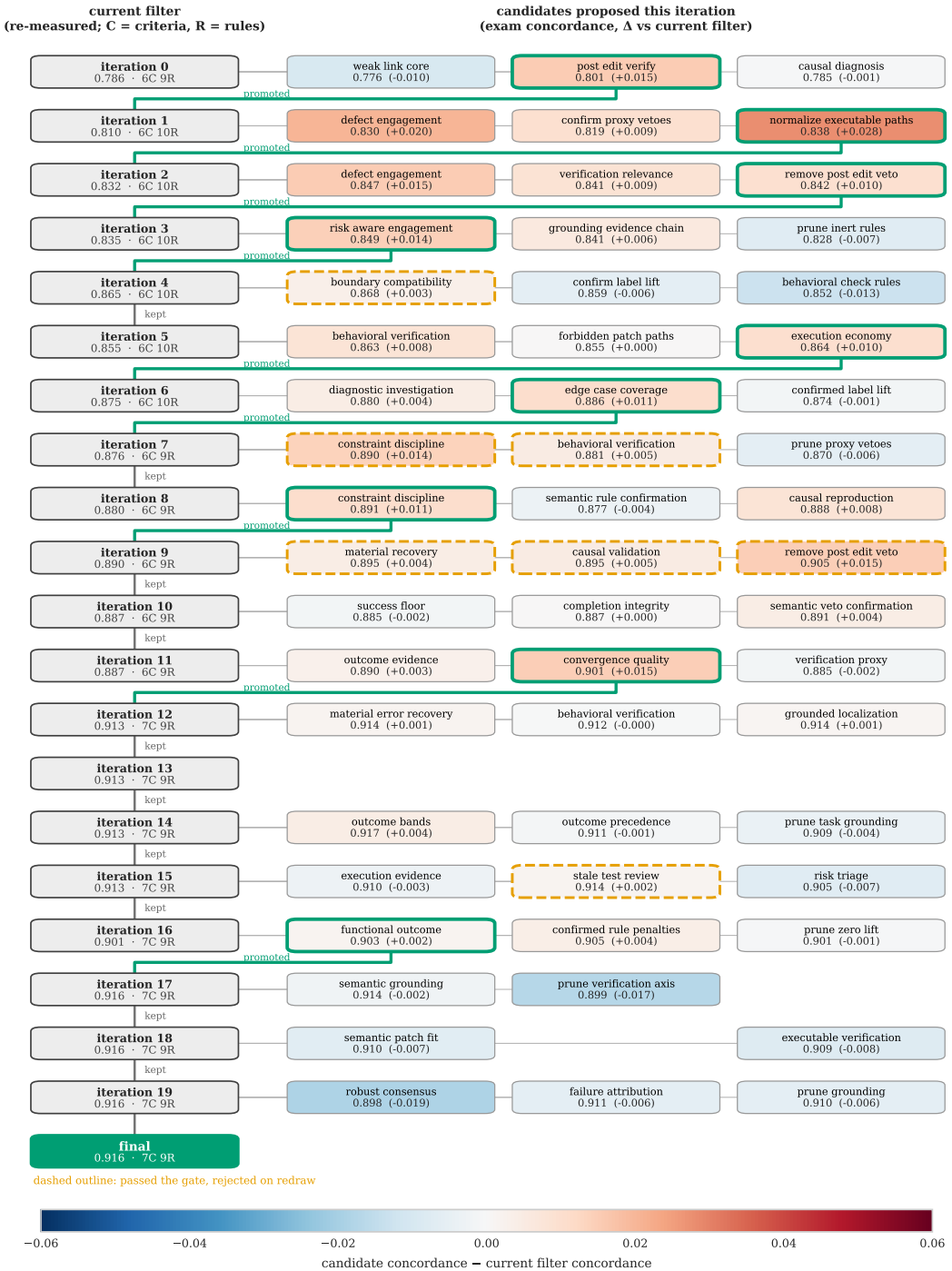

Figure 6: Search history from a handwritten seed over twenty iterations. Notation and color scale match Fig. 5. Mostly pale cells indicate few large departures from the current filter, consistent with greater inertia than in the blank-seed run.