How Much Reward Does Self-Evolution Need?

Pairwise Validation Without Task-Reward Feedback

Anonymous authors
Paper under double-blind review

Abstract

Self-evolving agents rely on task rewards to guide revisions, yet outside benchmarks labeled examples may be scarce, expert feedback costly, and automated evaluators incomplete. Can agents improve their prompts without reward feedback at every evolution step? We introduce PairEvolve, which removes task scores and gold answers from mutation and uses a frozen LLM to compare the original and revised outputs for acceptance. Two variants address different levels of reward access. PairEvolve-Val retains validation-score parent selection when a labeled set is available to choose which prompt to revise next. PairEvolve-Elo replaces that feedback with Elo ratings—numerical ratings updated from pairwise outcomes—so both acceptance and parent selection can proceed without task rewards. Both retain labeled final selection in our evaluation. Across four prompt tasks and two agent models, Val and Elo improve mean test scores over no evolution in five and six of eight settings, respectively, retaining full-reward performance in some settings and trading performance for reduced reward access in others. Validator studies identify how comparison format, thinking, and the treatment of undecided comparisons affect this feedback. Integrations with two program-evolution engines extend the comparison interface to code while retaining execution checks and proxy scores. Together, these results show how pairwise feedback can support evolution with fewer reward-dependent decisions.

1 Introduction

Self-evolving agent systems improve an artifact surrounding a frozen language model: a prompt, a program, or a workflow. A typical loop selects a parent artifact, proposes a child, evaluates both, and retains useful changes. Existing engines evolve prompts through reflective mutation and Pareto-based parent selection (Agrawal et al., 2025), programs (Cheng et al., 2025; Lange et al., 2025), and workflow graphs (Zhang et al., 2024). These systems optimize different objects, but each needs feedback to guide the next change.

A benchmark often supplies that feedback as a task reward: a score assigned by an evaluator to an artifact's output. For a new deployment task, labeled examples may be scarce, expert assessment costly, and a reliable automated evaluator difficult to build. A small labeled set may support occasional validation without supplying feedback for every proposed revision. Even an available metric may omit important behavior or create a target that optimization can exploit (Goodhart, 1984; Shao et al., 2025). Recent systems such as Jev (Almeida, 2026) separate model-based decisions from open-ended generation. For self-evolution, we ask which decisions can use comparative judgments instead of task-reward feedback.

We propose PairEvolve, which uses a frozen LLM to compare the outputs of a parent and its proposed child. The validator sees the task input and both outputs, without gold answers or task scores. Its judgment replaces reward-based acceptance, while the mutator reflects on generated outputs without task-score feedback.

The two variants differ in how they choose what to improve next. PairEvolve-Val retains validation-score parent selection: when a labeled set is available during evolution, it can guide which accepted prompts receive further revision without supervising each local proposal. PairEvolve-Elo instead uses Elo ratings, which summarize pairwise outcomes as numerical ratings for selecting parents. It is intended for evolution without validation-score feedback, reserving labeled evaluation for the final

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.

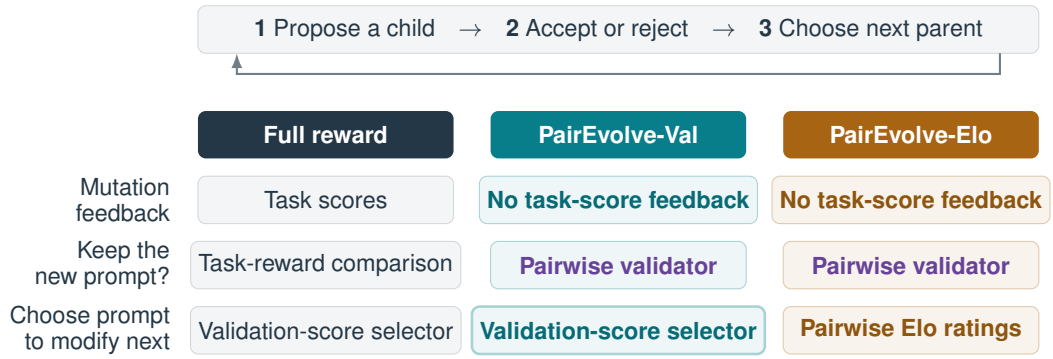


Figure 1: **Two uses of pairwise feedback.** Both variants replace reward-based acceptance with a **pairwise validator** and remove task-score feedback from mutation. Their difference is the next-parent rule: PairEvolve-Val retains validation-score selection; PairEvolve-Elo uses ratings from the pairwise comparisons. A child is a proposed revision of the selected parent prompt.

choice among the generated prompts. The distinction is where rewards remain available, not a claim that one variant is uniformly better. Both use labeled validation to select the final reported candidate.

Pairwise comparison fits the local decision an evolution engine must make: whether to retain a proposed change. It asks the validator to identify a better response, rather than assign two absolute scores on a common scale. The same comparisons can also guide parent selection. Preference feedback, LLM validators, and rating systems are established tools (Christiano et al., 2017; Bai et al., 2022; Zheng et al., 2023; Chiang et al., 2024). Here, we use them to replace specific sources of task-reward feedback within an existing evolution loop.

Figure 1 separates these decisions. Unlike the goal of end-to-end label-free prompt optimization (Wu et al., 2025), our question is which parts of a reward-guided engine can operate without task scores. We study a practical middle ground: evolution uses unlabeled task inputs, while a small labeled set remains available for parent or final selection. Success means improving the initial prompt with less reward feedback. Matching the full-reward reference is especially useful; exceeding it is not a requirement.

Our contributions are:

- **Agentic evolution with fewer reward-dependent decisions.** We introduce pairwise acceptance with validation-score or rating-based parent selection, and evaluate both configurations against no evolution and full reward. A multi-seed prompt study finds gains over no evolution, and two program engines extend the comparison interface beyond prompts.
- **Validator benchmarks for answer quality and presentation effects.** We construct fixed answer pairs to compare pairwise and pointwise evaluation and measure position and length preferences. The studies identify where direct comparison and thinking help, and where improved positional balance leaves quality errors unresolved.
- **A controlled analysis of acceptance rules.** We compare reward-based, pointwise, pairwise, and unconditional acceptance. When opposite answer orders yield conflicting preferences, recording a tie and accepting tied proposals can admit almost every revision. This explains how a validator can make comparisons yet exert little selection pressure.

2 Preliminary: Self-Evolution and Reward Feedback

Self-evolution improves **agent artifacts** used by a frozen model, rather than updating model weights. For example, each **candidate** in prompt evolution is a prompt; in program evolution, it is executable code. We describe the prompt setting first. Starting from a seed prompt f_0, the engine maintains a pool $\mathcal{P}_t$ of candidates. At iteration t, a **parent** $f_{t,p}$ is a prompt selected for modification, and its proposed revision $f_{t,c}$ is the **child**. We write $f(x)$ for the task model's output on input x under prompt f.

Outer loop: what to improve next. At iteration t, the engine selects a parent $f_{t,p} \in \mathcal{P}_t$, invokes a local proposal-and-comparison step, and adds the child if it passes the acceptance rule. Repeating this process produces a pool of evolved prompts. Parent selection determines which existing prompts receive further attempts at improvement; acceptance determines which new prompts become eligible. Algorithm 2 instantiates this outer loop and calls the pairwise inner step in Algorithm 1 once per iteration.

Inner step: propose and compare. A mutator $\mathcal{M}$ revises the parent using feedback h_t from a minibatch of task inputs:

$$f_{t,c} \sim \mathcal{M}(\cdot \mid f_{t,p}, h_t). \tag{1}$$

The feedback can include inputs, generated responses, and evaluator scores or comments. Parent and child are then compared on the same minibatch $\mathcal{T}_t \subset \mathcal{T}$. Here, the training pool $\mathcal{T}$ supplies evolution inputs, not model-weight updates; a separate validation pool $\mathcal{V}$ can guide parent selection.

Where reward enters. In the full-reward baseline (Agrawal et al., 2025), task rewards guide both loops. Within the inner step, evaluator feedback informs mutation, and the child is accepted when its minibatch reward improves on the parent's:

$$r_{\text{step}}(f_{t,c}; \mathcal{T}_t) > r_{\text{step}}(f_{t,p}; \mathcal{T}_t). \tag{2}$$

In the outer loop, a validation-based Pareto selector uses per-example scores to choose parents. It preserves prompts that perform well on different examples, rather than always choosing the highest average score. Thus, replacing reward-based acceptance alone does not remove reward feedback from mutation or parent selection. Figure 1 separates these three uses.

Our question is which of these decisions can instead use **pairwise feedback**. We use *reward-free* for a decision that receives neither externally supplied task-specific scores nor gold-answer feedback; it can still receive an LLM's comparative judgment.

3 PairEvolve

We propose PairEvolve, a method that uses **pairwise validation** to accept or reject revised agent artifacts. In prompt evolution, the inner step revises the parent $f_{t,p}$ without task scores or gold answers, then compares parent and child outputs (Algorithm 1). The outer loop retains accepted prompts and chooses the next parent (Algorithm 2). PairEvolve-Val uses validation scores for parent selection; PairEvolve-Elo uses pairwise ratings.

3.1 PairEvolve: Evolution through Pairwise Validation

Mutation without task-reward feedback. Algorithm 1 first runs the parent on $\mathcal{T}_t$. The mutator receives the inputs and responses, $h_t = \{(x, y_p(x)) : x \in \mathcal{T}_t\}$, and proposes $f_{t,c}$. It receives no informative task scores, evaluator comments, or gold answers. The change is to the information guiding mutation, not the host engine's prompt-revision mechanism.

Compare responses, not numerical scores. The child runs on the same inputs. The local question is whether its responses improve on the parent's, not what absolute score either response deserves. A frozen validator V_ϕ compares $y_c(x)$ with $y_p(x)$ and returns $d_x \in \{c, p, \text{tie}\}$, indicating a child win, parent win, or no preference between the two. It sees the task input and both responses, but no reference answer or task score. We randomize presentation order, map the decision back to parent/child labels, and request temperature zero. Algorithm 1 returns the child and the judgments $D_t = \{(x, d_x) : x \in \mathcal{T}_t\}$ to the outer loop.

Use the judgments for acceptance. The outer loop adds the child to $\mathcal{P}_t$ when its wins are at least the parent's:

$$\sum_{(x,d_x)\in D_t} \mathbf{1}[d_x = c] \geq \sum_{(x,d_x)\in D_t} \mathbf{1}[d_x = p]. \tag{3}$$

We use $|\mathcal{T}_t| = 3$. The non-strict inequality means tied aggregate decisions accept the child; this choice avoids halting evolution when the validator abstains, but it can weaken filtering (Section 5.2).

Algorithm 1: Pairwise step

Input: $f_{t,p}, \mathcal{T}_t, \mathcal{M}, V_\phi$.
Output: $f_{t,c}, D_t$.

$y_p(x) \leftarrow f_{t,p}(x), \quad \forall x \in \mathcal{T}_t$
$h_t \leftarrow \{(x, y_p(x)) : x \in \mathcal{T}_t\}$
$f_{t,c} \sim \mathcal{M}(\cdot \mid f_{t,p}, h_t)$
▷ Mutate without task scores or gold answers.
for each $x \in \mathcal{T}_t$ **do**
  $y_c(x) \leftarrow f_{t,c}(x)$
  $d_x \leftarrow V_\phi(x, y_c(x), y_p(x))$
  ▷ Validator compares parent and child outputs.
$D_t \leftarrow \{(x, d_x) : x \in \mathcal{T}_t\}$
return $(f_{t,c}, D_t)$

$d_x \in \{c, p, \text{tie}\}$; randomized output order.

Algorithm 2: Evolution loop

Input: $f_0, \mathcal{T}, \mathcal{V}$, budget T.
Output: evolved candidate pool $\mathcal{P}_T$.

$\mathcal{P}_0 \leftarrow \{f_0\}$; initialize ratings R (Elo)
for $t = 0, \ldots, T-1$ **do**; $\mathcal{P}_{t+1} \leftarrow \mathcal{P}_t$
  Val: $f_{t,p} \sim \text{ValSelect}(\mathcal{P}_t, \mathcal{V})$
  Elo: $f_{t,p} \sim \text{softmax}(R/\tau)$
  $\mathcal{T}_t \sim \text{Batch}(\mathcal{T})$
  $(f_{t,c}, D_t) \leftarrow$ **Algorithm 1**$(f_{t,p}, \mathcal{T}_t)$
  if $\text{Accept}(D_t)$ **then**
    ▷ Accept when child wins $\geq$ parent wins.
    $\mathcal{P}_{t+1} \leftarrow \mathcal{P}_t \cup \{f_{t,c}\}$
    Elo: $R \leftarrow \text{EloUpdate}(R, f_{t,p}, f_{t,c}, D_t)$
    ▷ Reuse validator decisions for parent selection.
return $\mathcal{P}_T$

Algorithms 1–2. The outer loop calls the pairwise inner step at iteration t; Val and Elo select the next parent. Acceptance and rating updates follow Eqs. 3 and 4.

The comparison reuses outputs from the training minibatch, without querying validation examples. Appendix A specifies input/output limits and the masking of evaluator outputs computed by the benchmark adapters. Program engines use the same local comparison interface with their own execution checks (Section 5.3).

3.2 Val and Elo: Two Parent-Selection Rules

An accepted child prompt $f_{t,c}$ joins the earlier prompts in the pool: $\mathcal{P}_{t+1} = \mathcal{P}_t \cup \{f_{t,c}\}$. The engine then selects the next parent $f_{t+1,p}$ from this entire pool, not necessarily the new child. Val uses labeled validation to choose that parent; Elo reuses pairwise outcomes when validation scores are unavailable to the evolution loop.

PairEvolve-Val: validation-score parent selection. Val retains the baseline's validation-based Pareto selector (Agrawal et al., 2025) for choosing among accepted prompts, while **pairwise validation** replaces reward feedback in the local step. The selector $\text{ValSelect}(\mathcal{P}_t, \mathcal{V})$ uses per-example validation scores to preserve prompts with different strengths. This is the teal branch in Algorithm 2.

PairEvolve-Elo: additionally remove validation-based parent selection. Elo replaces the validation-based selector with sampling from candidate ratings. Each accepted child receives a rating, and its existing judgments D_t update both child and parent. For each comparison,

$$\hat{S}_c = \frac{1}{1 + 10^{(R_o - R_c)/400}}, \qquad R_c \leftarrow R_c + k(S_c - \hat{S}_c), \tag{4}$$

where R_c and R_o are child and parent ratings, S_c is 1 for a child win, 0 for a loss, and $1/2$ for a tie, and k is the step size. The parent receives the opposite update. The gold branch in Algorithm 2 samples from $\text{softmax}(R/\tau)$ over $\mathcal{P}_t$, where τ is the sampling temperature. Higher ratings favor further mutation without excluding lower-rated prompts. Primary runs use only accepted parent–child judgments, without extra archive matches. These history-dependent ratings guide sampling, not absolute quality estimation; Appendix A gives the settings.

Validator configuration. The two variants also differ in the validator's **rubric**, the written criteria for comparing responses. Val uses an adaptive rubric: every ten iterations, the validator reviews recent decisions and output snippets and chooses a focus on correctness, correctness plus reasoning, or holistic quality. This is self-assessment without labeled evaluator feedback. Elo uses a fixed Direct rubric and unweighted rating updates. We therefore evaluate two complete configurations, not an isolated ablation of parent selection; Appendix A provides the rubric procedure.

Table 1: **PairEvolve can retain full-reward performance with less task-reward feedback.** Task scores (%), mean±sample SD over five Qwen3-8B seeds and three gpt-5.4-mini seeds. No Evolution uses the initial prompt; all evolved methods retain validation-based final selection.

Agent	Task	No Evolution	GEPA	PairEvolve (ours)	
			Full Reward	PairEvolve-Val	PairEvolve-Elo
gpt-5.4-mini	HotpotQA	29.2±1.5	77.1±2.9	84.0±2.2	79.8±5.9
	IFBench	84.8±2.2	90.4±2.0	90.4±2.1	89.3±2.7
	HoVer	58.0±2.0	62.7±1.2	50.0±3.5	52.0±11.1
	PUPA	58.7±1.7	73.1±3.9	68.1±12.1	65.3±5.2
Qwen3-8B	HotpotQA	63.7±2.3	70.7±8.7	69.4±7.6	66.0±6.2
	IFBench	92.1±2.4	86.1±3.0	88.4±2.7	85.0±4.4
	HoVer	45.6±3.8	49.6±10.9	44.0±13.3	53.6±7.8
	PUPA	53.6±1.0	71.3±6.7	57.9±4.3	59.7±3.8

4 Experimental Setup

Engines and models. Our prompt-evolution experiments use GEPA (Agrawal et al., 2025) as the shared engine for the full-reward baseline and both PairEvolve variants, with Qwen3-8B and gpt-5.4-mini as task models. In the primary comparison, the task model also serves as the validator. We preserve the engine's mutation mechanism, task evaluators, and iteration budgets while changing the reward feedback in Figure 1. Additional experiments study validator design and extend pairwise acceptance to program evolution in ADRS and ShinkaEvolve.

Tasks and metrics. We use four benchmark-derived tasks covering factual reasoning, explicit constraints, and privacy-preserving rewriting. HotpotQA measures multi-hop answer quality with token F1 (Yang et al., 2018). IFBench measures the fraction of supplied constraints satisfied, rather than strict all-constraints accuracy (Pyatkin et al., 2025). HoVer measures claim classification from claims and supporting-document pointers, without evidence text, using permissive label extraction (Jiang et al., 2020). PUPA combines reference-based rewriting quality with a penalty for personal-information leakage detected by string matching (Siyan et al., 2025). All methods within the primary comparison share these task adapters.

Evaluation. We compare both PairEvolve variants with the initial, unevolved prompt and with the full-reward baseline, which uses task rewards for mutation, acceptance, and parent selection. For each evolved run, validation scores select the reported prompt, whose performance is measured on held-out test examples. Appendix A specifies task metrics and adapters, data splits, evolution budgets, generation settings, and the checkpoint-selection procedure. Study-specific setups accompany the validator diagnostics and extension results.

5 Results and Analysis

We first ask whether PairEvolve improves the initial prompt and how much full-reward performance it retains (Section 5.1). We then examine the feedback itself: whether a validator recognizes better answers, resists presentation effects, and produces decisions that filter proposals (Section 5.2). Finally, we test the comparison interface in two program-evolution engines (Section 5.3). These studies distinguish the performance of a complete evolution system from the behavior of its validator.

5.1 Prompt Improvement with Less Task-Reward Feedback

PairEvolve-Val improves on no evolution in five of eight settings. Table 1 shows mean gains for both agents on HotpotQA and PUPA, and for gpt-5.4-mini on IFBench. The clearest result is gpt-5.4-mini HotpotQA, which rises from 29.2 to 84.0. Each of the three mini seeds improves over its unevolved reference on HotpotQA, IFBench, and PUPA, although the size of the PUPA gain varies considerably. Pairwise acceptance and validation-based selection thus support useful revisions without task scores guiding either mutation or acceptance.

Table 2: **Validator design: comparison format and thinking mode.** (a) Pairwise versus pointwise control agreement, estimate±SE; ties receive half credit and Δ is pairwise minus pointwise. (b/c) Thinking off versus on for position and length selection, mean±SD across three separate six-task banks. Position balance improves; length effects vary.

(a) Pairwise vs. pointwise agreement

Control agreement (%): higher is better

Task	Tier	Pairwise	Pointwise	Δ
DeepSeek V4 Flash (thinking on)				
HotpotQA	Clear	94.7±1.1	90.8±1.6	+3.9±1.3
HoVer	Clear	89.8±1.6	84.2±1.8	+5.6±1.9
HotpotQA	Moderate	88.5±1.8	86.2±1.8	+2.3±1.8
HoVer	Moderate	72.9±2.5	72.6±2.1	+0.2±2.4
HotpotQA	Near	69.0±2.3	68.3±2.2	+0.7±2.2
HoVer	Near	60.2±2.3	57.6±2.1	+2.6±2.5
Qwen3.8-27B (thinking on)				
HotpotQA	Clear	97.8±0.8	96.5±1.1	+1.4±0.7
HoVer	Clear	94.6±1.1	92.8±1.4	+1.8±1.2
HotpotQA	Moderate	91.0±1.7	90.4±1.6	+0.6±1.4
HoVer	Moderate	84.0±2.0	85.5±2.0	-1.5±1.8
HotpotQA	Near	84.6±1.9	76.6±2.3	+8.0±2.2
HoVer	Near	64.7±2.6	66.0±2.4	-1.3±2.7

(b) Thinking and position bias

First-position selection: 50% = balanced

Either extreme indicates more bias

Validator	Thinking off (%)	Thinking on (%)
Qwen3-8B	94.3±0.2	52.2±0.2
Gemma-4-26B	44.8±0.2	53.7±0.1
Qwen3.6-27B	71.1±0.2	50.1±0.3
Qwen3.5-122B	42.1±0.3	51.3±0.1
DeepSeek V4 Flash	96.2±0.2	53.0±0.2

(c) Thinking and length preference

Longer-output selection (%)

50% is a reference, not an accuracy target

Validator	Thinking off (%)	Thinking on (%)
Qwen3-8B	55.3±0.2	70.9±0.4
Gemma-4-26B	69.8±0.4	72.5±0.6
Qwen3.6-27B	61.4±0.7	58.4±0.1
Qwen3.5-122B	67.4±0.7	59.0±0.4
DeepSeek V4 Flash	54.1±0.3	61.5±0.9

PairEvolve-Elo improves on no evolution in six of eight settings. Elo also improves all three mini seeds on HotpotQA, IFBench, and PUPA. With Qwen3-8B, its mean exceeds no evolution on HotpotQA, HoVer, and PUPA; the PUPA gain holds across all five seeds, while the other two tasks vary more across runs. Here, task rewards guide neither local proposals nor the choice of the next parent. Labeled validation is used to choose the returned prompt, showing that useful candidates can be produced before that final reward-based decision.

PairEvolve can retain full-reward performance. The full-reward comparison asks how much performance is retained when feedback is removed. On gpt-5.4-mini HotpotQA, both variants exceed the baseline's mean: 84.0 for Val and 79.8 for Elo versus 77.1. Val also matches its observed mean of 90.4 on mini IFBench, and Qwen3-8B HotpotQA reaches 69.4 with Val versus 70.7 with full reward. PUPA shows the tradeoff more clearly: both variants improve over no evolution for both agents, but retaining all reward feedback yields higher scores. The task reward combines reference-based quality and leakage checks that an output-only comparison does not directly observe.

The two references also distinguish useful gains from relative rankings. Qwen3-8B Val exceeds the full-reward baseline on IFBench but remains below no evolution, and neither variant improves the mean score on mini HoVer. Thus, reduced reward access is useful in several settings, not a guarantee that evolution helps on every task. The positive results establish the complete configurations; the next study examines the quality of the comparisons they rely on. Appendix B reports every seed and the common final-selection rule.

5.2 Validator Ablations and Diagnostics

A validator has to do more than return a preference. It must recognize a useful change, avoid choosing by presentation alone, and supply judgments that lead to meaningful acceptance decisions. We study the first two requirements on fixed answer pairs, so differences between validators cannot be attributed to different proposed prompts. A controlled evolution study then examines how the acceptance rule uses those judgments. Together, these tests explain what to check before putting a pairwise validator into an evolution loop.

5.2.1 Can Validators Recognize Better Answers?

A prompt revision can change an answer's correctness, its reasoning, or just its presentation. To test whether validators recognize the difference, we construct fixed answer pairs with a known intended change. Claude Opus 5 uses source questions, reference answers, and evaluator rules to write a **control answer** and an alternative with one annotated flaw. The control is the intended better answer. Validators receive only the task input and the two answers, without references or flaw annotations.

The benchmark contains 1,370 pairs from 457 HotpotQA and HoVer questions. Three **difficulty tiers** probe how much checking a comparison requires: *clear* errors are apparent from the response, *moderate* errors involve an invalid inference, and *near* errors require factual verification or recomputation. This separates spotting obvious defects from distinguishing two plausible answers, a capability relevant when a prompt revision changes the response only slightly. We screen generated pairs using identifier, formatting, and evaluator checks. Appendix D gives the construction procedure, checks, and annotation limitations.

Control agreement measures how often a validator favors the designated better answer, with half credit for a tie. Comparing agreement across tiers reveals where a validator can resolve a quality difference and where the distinction becomes difficult. Table 2a compares pairwise and pointwise evaluation on these pairs. Panels b/c use a separate set of three six-task banks, with identical pairs across thinking modes. This sweep includes Qwen3.6-27B rather than panel a's Qwen3.8-27B. Appendix D.2 reports the full five-model results, pooled and by task.

5.2.2 Pairwise Comparison versus Pointwise Scoring

A **pairwise validator** sees both answers together and chooses the better one or declares a tie. A **pointwise validator** sees each answer separately and assigns it a numerical score; comparing those scores produces the decision. Acceptance is a relative question: direct comparison can focus on the change, whereas two independent scores must resolve it on a common scale. Table 2a compares these formats on the same valid records. Pairwise decisions reconcile both presentation orders, treating disagreement as a tie; equal pointwise scores are also ties. Both receive half credit for ties.

Pairwise comparison helps most clearly when the answer difference is identifiable. Pairwise agreement is higher in all four clear-error comparisons in Table 2a. The differences are larger for DeepSeek than for Qwen, whose pointwise scores already separate most clear errors. For subtler changes, the advantage depends on the task and validator, and several differences are small relative to their standard errors. This supports pairwise comparison as a natural starting point for acceptance: it asks directly which response is better, without requiring two independently assigned scores to resolve the change. It does not remove the need for task evidence when both answers are plausible.

5.2.3 Thinking, Position Bias, and Length Preference

Position bias is sensitivity to whether an answer is displayed first or second, rather than to its content. We evaluate each pair in both orders and report *first-position selection*: the fraction of decided raw calls choosing the first answer. Its balanced value is 50%; values toward either 0% or 100% indicate a positional preference. This is distinct from control agreement: a validator can recognize many clear errors while still favoring one position.

Thinking-enabled configurations reduce aggregate position bias. Table 2b compares five validators across three banks covering HotpotQA, HoVer, IFBench, PUPA, AIME, and LiveBench-Math. Thinking on brings pooled first-position selection closer to balance for every validator. Without thinking, Qwen3-8B and DeepSeek almost always choose the first answer. Randomizing order removes a fixed advantage for the parent or child, but cannot turn such choices into quality judgments. Thinking with a larger output allowance addresses this failure much more effectively in this sweep; the comparison changes both mode and allowance.

Position balance is one check, not the whole decision. Table 2c tracks selection of the longer answer. With thinking, Qwen3.6-27B and Qwen3.5-122B combine higher clear-control agreement with improved position and length balance; the other validators choose longer answers more often. Unlike answer position, length can carry useful information, so 50% is a descriptive reference rather than an accuracy target. The relevant question is whether the validator prefers an answer for its content. This is why task-specific agreement should be checked alongside both presentation measures.

Table 3: **Pairwise acceptance in program evolution.** Terminal scores from one run per configuration; higher is better within rows. Pairwise + checks retains execution checks and proxy scores.

Engine	Task	Objective	Full reward	Execution checks	Pairwise + checks (ours)
ADRS	Circle packing	Radii sum ↑	1.76	1.84	2.16
	Scheduling	Inv. makespan ↑	3787.9	3257.3	3174.6
ShinkaEvolve	Circle packing	Radii sum ↑	1.37	1.39	1.90
	Prime counting	Accuracy–runtime ↑	99.63	99.73	99.74

PUPA illustrates the distinction: position balance improves, yet clear-control agreement remains near or below chance (Appendix D.2). Together with the full-reward advantage on PUPA, this points to reference-dependent quality as a difficult case for substitution, though the two studies use different configurations.

Tie handling determines whether comparisons filter proposals. In a HotpotQA/HoVer study, Qwen3.8-27B generates and DeepSeek V4 Flash validates, with thinking requested off. Full reward, pointwise, symmetric pairwise, and no validator share validation-based parent and final selection; no validator admits every proposal. Judging both answer orders and turning disagreements into accepted ties admits over 99% of proposals, with endpoint scores close to no validator (Appendix C). Reconciling presentation order can therefore leave almost all selection to later stages. The primary Val/Elo rules filter more proposals but use different configurations (Appendix B.1). Acceptance records show how the judgments are used, complementing output-level accuracy and bias diagnostics.

5.3 Extension to Program-Evolution Engines

ADRS and ShinkaEvolve evolve programs by proposing code changes, executing the resulting programs, and retaining candidates for further revision. We use Qwen3-8B to generate code and Haiku 4.5 as the frozen pairwise validator.

Three feedback configurations. Table 3 compares *Full reward*, the engine's task-score-guided reference; *Execution checks*, which uses programmatic checks without an LLM validator; and our *Pairwise + checks*, which adds a parent–child comparison after execution and regression checks. The pairwise validator sees both programs and the outputs and proxy scores from their execution. Thus, the added decision is whether the proposed code revision should be accepted; execution checks and proxy scores remain available.

Tasks and scores. Circle packing places non-overlapping circles inside a fixed container and maximizes their total radii. Scheduling minimizes the time to finish all jobs (makespan), reported as an inverse score so larger is better. Prime counting combines correctness and runtime in one score, not an accuracy percentage. Scores are compared within each engine–task row.

Pairwise validation also supports executable artifacts. On circle packing, Pairwise + checks improves over both references: it reaches 2.16 versus 1.76 for full reward in ADRS, and 1.90 versus 1.37 in ShinkaEvolve. Scheduling favors full reward, while prime-counting scores are closely grouped. The shared acceptance decision can therefore use different evidence across artifact types: generated answers for prompts, and program behavior for code. Appendix E gives the complete configurations, score definitions, and additional signal-processing results.

6 Related Work

Pairwise feedback and LLM judges. Human and AI preferences are widely used to learn reward models or select outputs (Christiano et al., 2017; Bai et al., 2022; Zheng et al., 2023). Chatbot Arena aggregates pairwise outcomes into ratings (Chiang et al., 2024). Quality-diversity methods have also incorporated AI or human feedback (Bradley et al., 2023; Ding et al., 2024), and preference-based evolutionary multi-objective optimization uses comparisons in place of scalar fitness (Huang et al., 2024). Elo-Evolve uses competitive outcomes and Elo-based opponent selection for model alignment training (Zhao et al., 2026); we instead evolve prompts or programs with frozen model weights.

Confidence and decision models. ConfTuner (Li et al., 2025) learns verbalized confidence from answer-correctness labels, with applications in self-correction and model cascades. Jev (Almeida, 2026) exposes bounded choices and scores for software decisions, while the independent JevTuner implementation (Li, 2026) trains distributions over supplied choices with a multiclass Brier loss. These efforts develop confidence estimates and decision interfaces; PairEvolve studies where frozen pairwise judgments can replace task-reward feedback within an evolution loop.

Self-evolving agents and rewardless search. GEPA, ADRS, ShinkaEvolve, AFlow, AlphaEvolve, OpenEvolve, and the Darwin Gödel Machine search over prompts, programs, workflows, or agent code (Agrawal et al., 2025; Cheng et al., 2025; Lange et al., 2025; Zhang et al., 2024; Google DeepMind, 2025; OpenEvolve Contributors, 2025; Zhang et al., 2026). CodeFavor and ExPairT-LLM study code preference judgments (Liu et al., 2025; Yuviler & Drachsler-Cohen, 2026); ScoreFlow uses preference information in workflow optimization (Wang et al., 2025). MADE uses decomposed LLM judging for evolution without an objective oracle (Zhao et al., 2025). Duel-Evolve uses a Bayesian Bradley–Terry model of self-preferences for test-time output search (Karlekar et al., 2026).

Prompt Duel Optimizer (PDO) performs label-free prompt optimization using pairwise output comparisons, dueling-bandit selection, and mutation of strong prompts, and studies pairwise versus pointwise judging (Wu et al., 2025). Our study focuses on substituting particular reward pathways inside existing evolution engines: local parent–child comparisons govern acceptance, and Val/Elo configurations vary the retained selection feedback.

Feedback Descent combines pairwise preferences with critiques and uses correctness and evaluation feedback in its prompt-optimization setting (Lee et al., 2025). Pairwise feedback is therefore not necessarily reward-independent. Unlike critique-driven mutation, our primary validator does not pass judge rationales to the mutator. Trace-level work further distinguishes mechanisms behind code-score gains (Pelleriti et al., 2026); our code endpoints establish interface scope, not algorithmic novelty.

Multi-objective optimization. Pareto methods such as NSGA-II and MOEA/D retain tradeoffs among scalar objectives (Deb et al., 2002; Zhang & Li, 2007). Pairwise acceptance answers a different question: whether a child enters the candidate pool. In PairEvolve-Val, the two operate at complementary stages: pairwise comparisons determine entry, and Pareto-based selection chooses which retained prompts to mutate next.

7 Limitations and Conclusion

The primary study uses retrospective configurations, small test sets, adapter-specific metrics, and labeled final selection. The diagnostics measure agreement with constructed preferences, not independently verified correctness for every pair. Program integrations retain proxy scores and have one run per configuration. These choices support conclusions about the evaluated systems; larger evaluations and label-free final selection remain open directions.

Agents can improve their prompts without task rewards guiding every evolution decision. PairEvolve replaces reward feedback in mutation and acceptance; Val retains labeled guidance for parent selection, while Elo uses pairwise ratings to reserve that guidance for final selection. The experiments show both retained full-reward performance and task-dependent tradeoffs. They also show why choosing a validator is only part of the design: it must recognize relevant answer differences, resist presentation effects, and interact sensibly with the acceptance rule. Pairwise feedback offers a way to reserve task rewards for fewer decisions while continuing to improve the artifact.

AI Use Statement

AI tools assisted with research, manuscript preparation, and synthetic diagnostic data generation. The authors reviewed their outputs and take responsibility for the work.

Ethics Statement

Privacy-rewriting scores are task metrics, not guarantees that sensitive information is protected. LLM judges can inherit capability gaps and biases, and pairwise acceptance is not a safety certificate. Evolved programs require isolated execution and independent checks before practical use. Benchmark outcomes do not establish safe autonomous deployment beyond the evaluated settings.

Reproducibility Statement

Sections 3–4 and the appendix specify configurations, reward pathways, analysis rules, budgets, and evolved examples. Saved checkpoints and evaluations support the reported results; incomplete request and execution logs limit exact replay.

References

Lakshya A Agrawal et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Diogo Almeida. Introducing system one models & Jev. TypeSafe AI Blog, 2026. URL `https://typesafe.ai/blog/introducing-system-one-models-and-jev`. September 15, 2026.

Yuntao Bai et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Herbie Bradley, Andrew Dai, Hannah Teufel, Jenny Zhang, Koen Oostermeijer, Marco Bellagente, Jeff Clune, Kenneth Stanley, Grégory Schott, and Joel Lehman. Quality-diversity through AI feedback. *arXiv preprint arXiv:2310.13032*, 2023.

Audrey Cheng et al. Barbarians at the gate: How AI is upending systems research. *arXiv preprint arXiv:2510.06189*, 2025.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael I. Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating LLMs by human preference. In *ICML*, 2024.

Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *NeurIPS*, 2017.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

Li Ding, Jenny Zhang, Jeff Clune, Lee Spector, and Joel Lehman. Quality diversity through human feedback: Towards open-ended diversity-driven optimization. In *ICML*, 2024.

Charles Goodhart. Problems of monetary management: The UK experience. In *Monetary Theory and Practice*. Macmillan, 1984.

Google DeepMind. AlphaEvolve: A Gemini-powered coding agent for designing advanced algorithms, May 2025. URL `https://deepmind.google/blog/alphaevolve-a-gemini-powered-coding-agent-for-designing-advanced-algorithms/`.

Tian Huang, Shengbo Wang, and Ke Li. Direct preference-based evolutionary multi-objective optimization with dueling bandits. In *NeurIPS*, 2024.

Yichen Jiang, Shikha Bordia, Zheng Zhong, Charles Dognin, Maneesh Singh, and Mohit Bansal. HoVer: A dataset for many-hop fact extraction and claim verification. In *Findings of EMNLP*, 2020.

Sweta Karlekar, Carolina Zheng, Magnus Saebo, Nicolas Beltran-Velez, Shuyang Yu, John Bowlan, Michal Kucer, and David Blei. Duel-evolve: Reward-free test-time scaling via LLM self-preferences. *arXiv preprint arXiv:2602.21585*, 2026.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

Yoonho Lee, Joseph Boen, and Chelsea Finn. Feedback descent: Open-ended text optimization via pairwise comparison. *arXiv preprint arXiv:2511.07919*, 2025.

Yibo Li. JevTuner. GitHub repository, 2026. URL https://github.com/liushiliushi/JevTuner. Accessed September 21, 2026.

Yibo Li, Miao Xiong, Jiaying Wu, and Bryan Hooi. ConfTuner: Training large language models to express their confidence verbally. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://arxiv.org/abs/2508.18847.

Jiawei Liu, Thanh Nguyen, Mingyue Shang, Hantian Ding, Xiaopeng Li, Yu Yu, Varun Kumar, and Zijian Wang. Do llms understand code preference? training code preference models via synthetic code evolution. In *DL4C Workshop, ICLR*, 2025.

OpenEvolve Contributors. OpenEvolve: An open-source evolutionary coding agent. GitHub repository, 2025. URL https://github.com/codelion/openevolve.

Nico Pelleriti, Sree Harsha Nelaturu, Zhanke Zhou, Zongze Li, Max Zimmer, Bo Han, and Sebastian Pokutta. What do evolutionary coding agents evolve? *arXiv preprint arXiv:2605.20086*, 2026.

Valentina Pyatkin et al. Generalizing verifiable instruction following. In *NeurIPS Datasets and Benchmarks Track*, 2025.

Shuai Shao, Qihan Ren, Chen Qian, Boyi Wei, Dadi Guo, Jingyi Yang, Xinhao Song, Linfeng Zhang, Weinan Zhang, Dongrui Liu, and Jing Shao. Your agent may misevolve: Emergent risks in self-evolving LLM agents. *arXiv preprint arXiv:2509.26354*, 2025.

Li Siyan, Vethavikashini Chithrra Raghuram, Omar Khattab, Julia Hirschberg, and Zhou Yu. PAPILLON: Privacy preservation from internet-based and local language model ensembles. In *Proceedings of NAACL: Human Language Technologies (Volume 1: Long Papers)*, pp. 3371–3390, 2025. doi: 10.18653/v1/2025.naacl-long.173. URL https://aclanthology.org/2025.naacl-long.173/.

Yinjie Wang et al. ScoreFlow: Mastering LLM agent workflows via score-based preference optimization. *arXiv preprint arXiv:2502.04306*, 2025.

Colin White, Samuel Dooley, et al. LiveBench: A challenging, contamination-limited LLM benchmark. arXiv preprint arXiv:2406.19314, 2024.

Yuanchen Wu, Saurabh Verma, Justin Lee, Fangzhou Xiong, Poppy Zhang, Amel Awadelkarim, Xu Chen, Yubai Yuan, and Shawndra Hill. Llm prompt duel optimizer: Efficient label-free prompt optimization. *arXiv preprint arXiv:2510.13907*, 2025. URL https://arxiv.org/abs/2510.13907.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*, 2018.

Tom Yuviler and Dana Drachsler-Cohen. ExPairT-LLM: Exact learning for LLM code selection by pairwise queries. In *AAAI*, 2026.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents. In *ICLR*, 2026.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui 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. *arXiv preprint arXiv:2410.10762*, 2024.

Qingfu Zhang and Hui Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

Jing Zhao, Ting Zhen, Junwei Bao, Hongfei Jiang, and Yang Song. Elo-Evolve: A co-evolutionary framework for language model alignment. *arXiv preprint arXiv:2602.13575*, 2026.

Zhe Zhao, Yuheng Yang, Haibin Wen, Xiaojie Qiu, Zaixi Zhang, and Qingfu Zhang. Evolution without an oracle: Driving effective evolution with LLM judges. *arXiv preprint arXiv:2511.19489*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *NeurIPS*, 2023.

APPENDIX

A IMPLEMENTATION DETAILS AND REWARD PATHWAYS

The studies answer different questions and use different model configurations. Table 4 summarizes their roles before we describe the protocols. Appendices B–C report evolution outcomes and acceptance behavior; Appendix D examines validator design. Program integrations, budgets, and artifact examples follow in Appendices E–G.

Table 4: **Questions and configurations across the studies.** Primary and controlled evolution evaluate returned prompts. Fixed-output diagnostics test the comparisons themselves; the remaining studies change the validator or the host engine.

Study	Models and protocol	Question
Primary evolution	Qwen3-8B or gpt-5.4-mini generates and self-validates. Val uses validation-based parents; Elo uses ratings. Both retain labeled final selection.	Can prompts improve with fewer reward-dependent decisions?
Comparison format	DeepSeek V4 Flash and Qwen3.8-27B, thinking on; pairwise and pointwise judgments of the same HotpotQA/HoVer outputs.	Which format better distinguishes answer quality?
Thinking and bias	Five validators, thinking off/on, on three six-task output banks. Generation allowances differ between modes.	How do mode and presentation affect decisions?
Controlled acceptance	Qwen3.8-27B generates; DeepSeek V4 Flash validates, with thinking requested off. Acceptance rules share validation-based parent and final selection.	How do comparisons and tie policy affect filtering and endpoint scores?
Validator replacement	Qwen3-8B generates; itself or Haiku 4.5 validates fresh outputs on validation inputs without their labels.	Can generation and validation use different models?
Program evolution	Qwen3-8B generates; Haiku 4.5 validates in ADRS and ShinkaEvolve. Execution checks and proxy scores remain available.	Can the same acceptance interface compare executable artifacts?

A.1 METRICS, DATA, AND STUDY INCLUSION

Metric definitions. Scores average example-level values and multiply by 100. HotpotQA removes thinking blocks, extracts the answer after a recognized prefix when present, lowercases, removes punctuation and articles, and computes multiset token F1 against the reference. Empty answers or zero overlap score zero; supporting facts are not scored.

IFBench averages the fraction of supplied constraints passed per example, not the official strict all-constraints-pass metric. Failed `check_following` execution can fall through to string or keyword checks; unverifiable dictionary constraints default to passing. Without constraints, a nonempty response scores one and an empty response zero. Fallback frequencies are unavailable.

PUPA equally weights rewriting quality and one minus leakage. Leakage is the fraction of supplied PII strings found as case-insensitive response substrings, or zero if none are supplied. A quality judge using the task-model alias sees the redacted reference and candidate. Its first unsigned number is clipped to $[0, 1]$; exceptions or unparseable responses default to 0.5, with fallback frequencies unrecorded. This combines reference-based LLM judgment with literal leakage detection, not a privacy guarantee.

HoVer supplies claims and oracle supporting-document titles/sentence indices, but no evidence text: this is restricted-input classification, not full retrieval and verification. The evaluator checks for `NOT_SUPPORTED`, then `SUPPORTED`, by substring. All primary arms share this rule, which can credit invented labels containing those strings rather than measure final-label accuracy (Appendix D.4).

The available records do not establish how often these fallback and parsing behaviors affect the primary test scores. The reported results use the adapters as run; they are not estimates under stricter replacement metrics.

Data and splits. Sources are HotpotQA's distractor validation split, IFBench's training split, PUPA's `pupa_tnb` training configuration, and HoVer's training release. The loader shuffles cached rows with split seed 42 (distinct from evolution seeds), truncates, and takes consecutive train/validation/test slices. Training sizes are 150 for HotpotQA, HoVer, and IFBench, and 111 for PUPA; validation sizes are 30. Test sizes are 50 for HotpotQA/HoVer and 30 for IFBench/PUPA. Row positions are disjoint; duplicate content and exact ordering were not independently verified.

Budgets and checkpoint selection. Iteration budgets are 350 for HotpotQA, 375 for HoVer, 200 for IFBench, and 125 for PUPA. For each evolved trajectory, we report the first test evaluation associated with its earliest maximum-validation candidate. The full-reward trajectory's iteration-zero evaluation supplies the unevolved seed reference.

Study inclusion. All 96 runs of the retrospectively specified primary configurations have validation-selected results: five Qwen3-8B seeds and three gpt-5.4-mini seeds across four tasks and three methods. An incomplete two-seed mini extension is excluded as a whole; other configurations appear separately as ablations.

Generation settings. Saved configurations use one model for generation, reflection, and self-validation, but omit task/reflection temperatures, token limits, and the primary thinking mode. The available implementation requests judge temperature zero, a 512-token judge limit, and a 256-token rubric-self-assessment limit; incomplete request logs prevent confirming these settings for every call. The thinking-on/off diagnostics are separate experiments with specified modes, not reconstructions of the primary serving settings.

Validator inputs and adaptive rubric. All adapters present a question and responses A/B, requesting `A_BETTER`, `B_BETTER`, or `TIE` on the first line. Direct criteria are reasoning, completeness, consistency, and clarity. Input/output prefixes are capped at 2,000/3,000 characters; input truncation rates are unavailable. Adaptive judging starts with correctness. Every ten iterations, the same judge receives recent parent/child win counts and acceptance decisions, plus up to three output pairs as 200-character snippets. It is asked which quality dimension most needs improvement and must return `correctness`, `correctness+reasoning`, or `full` on its first line. The parser strips whitespace and lowercases that line; a recognized keyword selects the corresponding predefined rubric, while invalid responses or request failures retain the current focus. No gold answers or evaluator scores enter this update.

Table 5: **Feedback and selection signals.** Earlier pointwise selects by absolute scores; the controlled study instead fixes validation-based selection.

Configuration	Mutation feedback	Acceptance	Parent selection	Final selection
Full reward	Outputs + rewards	Task reward	Validation	Validation
PairEvolve-Val	Outputs only	Pairwise	Validation	Validation
PairEvolve-Elo	Outputs only	Pairwise	Elo	Validation
Pointwise (earlier)	Outputs only	Absolute scores	Absolute scores	Absolute scores

A.2 Mutation and Acceptance

Evolution seeds are $\{42, 50, 51, 52, 53\}$ for Qwen3-8B and $\{42, 50, 51\}$ for gpt-5.4-mini. Both methods compare three training examples per proposal and randomize parent/child placement independently. Val uses adaptive judging and validation-based Pareto parent selection; Elo uses Direct judging without added context or warmup, with the rating settings in Figure 2. Missing outputs and failed judge calls become ties; an all-tie proposal is accepted under Eq. 3.

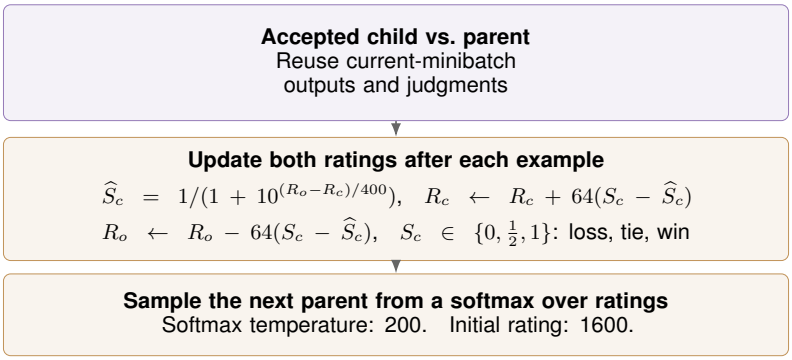


Figure 2: **PairEvolve-Elo pipeline.** Accepted parent–child comparisons update unweighted ratings for future parent selection. The primary runs reuse acceptance judgments without additional archive opponents or inference calls.

The proposer adapter replaces training scores by 0.5, substitutes a fixed self-assessment instruction for evaluator feedback, and removes gold answers and auxiliary evaluator context. Reflection receives only task inputs, generated responses, and that instruction. A wrapper bypasses the native reward-improvement pre-check and exposes current-minibatch parent/child outputs to the pairwise validator before candidate commitment. Accepted Val children receive validation scores and enter the pool for Pareto-based parent selection; Elo comparisons update ratings for parent sampling.

The shared `accept_or_reject(parent, child)` interface also supports the code integrations (Appendix E). Evaluators execute before masking: reward information is removed from evolution decisions, not from evaluator computation.

A.3 PairEvolve-Elo Mechanics

The primary runs reuse the three acceptance judgments, with no additional archive opponents ($E = 0$) or inference calls. Only accepted proposals update both ratings, sequentially by example. Ratings start at 1600, use $k = 64$, and give parent-sampling probability proportional to $\exp((R - \max R)/200)$, without warmup (Figure 2).

Ratings are a sampling heuristic, not calibrated absolute quality estimates: fresh-child initialization, encounter history, and example order affect them, and parents receive no credit for rejecting inferior children. The results evaluate this complete rule rather than isolate its design choices.

B Primary Results and Acceptance Behavior

Table 6 contains all 96 evolved endpoints and 32 seed references underlying Table 1. Evolved results use the common validation-based checkpoint rule; seed references reuse the full-reward trajectory's iteration-zero evaluation. Matching configured seeds pairs trials without implying identical generation or serving randomness. Appendix A defines the task-specific metrics.

Table 6: **Primary results by seed.** Task scores (%); all evolved methods use validation-based final selection. The seed prompt is the full-reward trajectory's iteration-zero reference.

Agent	Task	Seed	No Evolution	Full Reward	PairEvolve-Val	PairEvolve-Elo
Qwen3-8B	HotpotQA	42	62.8	75.2	63.7	71.2
		50	60.1	73.9	71.6	57.8
		51	64.7	73.3	72.4	65.5
		52	66.1	55.2	79.3	72.9
		53	64.6	75.7	60.2	62.8

Table 6: Primary results by seed (continued).

Agent	Task	Seed	No Evolution	Full Reward	PairEvolve-Val	PairEvolve-Elo
Qwen3-8B	IFBench	42	91.1	83.9	85.0	86.1
		50	94.4	82.2	88.3	79.4
		51	88.9	88.9	90.6	91.7
		52	91.7	86.7	86.7	83.9
		53	94.4	88.9	91.7	83.9
Qwen3-8B	HoVer	42	40.0	64.0	28.0	60.0
		50	46.0	52.0	62.0	48.0
		51	44.0	42.0	42.0	64.0
		52	50.0	36.0	52.0	48.0
		53	48.0	54.0	36.0	48.0
Qwen3-8B	PUPA	42	52.4	67.1	61.1	53.7
		50	53.7	62.7	57.8	58.5
		51	53.1	74.6	62.6	62.1
		52	54.0	72.3	56.7	60.9
		53	54.9	80.0	51.5	63.2
gpt-5.4-mini	HotpotQA	42	30.9	75.2	85.8	73.0
		50	28.9	75.7	81.5	82.7
		51	27.8	80.5	84.6	83.6
gpt-5.4-mini	IFBench	42	87.2	90.6	89.4	91.1
		50	84.4	92.2	92.8	90.6
		51	82.8	88.3	88.9	86.1
gpt-5.4-mini	HoVer	42	56.0	62.0	48.0	54.0
		50	60.0	62.0	54.0	40.0
		51	58.0	64.0	48.0	62.0
gpt-5.4-mini	PUPA	42	57.3	70.8	60.6	70.7
		50	58.2	77.6	61.6	60.3
		51	60.6	70.9	82.1	64.9

B.1 Acceptance in the Primary Runs

Table 7 reports cumulative accepted and rejected decisions for all 64 Val/Elo trajectories underlying Table 1. Rates pool decisions across seeds within each task; they are not seed-mean accuracies. The counters are monotonic and agree with their terminal values. Fourteen Qwen3-8B trajectories have one fewer logged decision than configured iterations (five Val, nine Elo); these unaccounted iterations are excluded rather than classified as rejections or ties. Detailed verdict records are absent, so semantic ties, failed judgments, and order effects cannot be separated for these runs.

Table 7: **Primary acceptance behavior.** Accepted/logged decisions and pooled acceptance rates. Each task combines five Qwen3-8B or three gpt-5.4-mini seeds.

	PairEvolve-Val		PairEvolve-Elo	
Task	Accepted / logged	Rate (%)	Accepted / logged	Rate (%)
Qwen3-8B				
HotpotQA	907 / 1750	51.8	907 / 1750	51.8
IFBench	519 / 997	52.1	512 / 995	51.5
HoVer	950 / 1875	50.7	952 / 1874	50.8
PUPA	347 / 623	55.7	334 / 622	53.7
gpt-5.4-mini				
HotpotQA	708 / 1050	67.4	682 / 1050	65.0
IFBench	195 / 600	32.5	280 / 600	46.7
HoVer	749 / 1125	66.6	701 / 1125	62.3
PUPA	179 / 375	47.7	142 / 375	37.9

The primary rules reject a substantial fraction of logged proposals, unlike the near-pass-through symmetric rule in the controlled study. Because agents, tasks, and comparator settings differ, this is a description of filtering behavior, not a matched estimate of the benefit of filtering. The controlled study separately examines endpoint performance with shared validation-based selectors.

B.2 Final Candidate Selection

All primary configurations return the candidate with the highest validation score, breaking ties in favor of the earliest candidate. For PairEvolve-Elo, validation scores select this final artifact but do not select parents during evolution. The reported scores therefore evaluate a common labeled final-selection rule.

C Controlled Comparison of Acceptance Rules

This study compares acceptance rules while holding validation-based parent and final selection fixed. Three arms share reward-blind mutation: no validator admits every proposal, pointwise compares independent output scores, and symmetric pairwise reconciles comparisons in both orders. Full reward supplies a reward-guided reference. The three reward-blind arms isolate the choice of acceptance rule; their endpoint scores and acceptance records show how that rule interacts with the shared downstream selectors. These configurations differ from the primary Val/Elo variants.

C.1 Experimental Settings

Qwen3.8-27B generates and reflects on task outputs, and DeepSeek V4 Flash provides the frozen judgments. We run HotpotQA and HoVer with seeds $\{42, 50, 51\}$, using 150 training, 30 validation, and 50 held-out examples per task. Training uses 350 HotpotQA or 375 HoVer iterations and three examples per proposal. The data pools are fixed across configurations; shared seeds identify paired trials, while their evolving populations and subsequent proposals can differ.

HoVer inputs include the claim and its annotation-provided evidence documents. The evaluator requires a recognized label on the final line; malformed labels receive zero. This is classification with provided documents. HotpotQA uses normalized answer-token F1. These settings differ from the restricted-input HoVer and primary agents in Table 1.

The four configurations are:

- **Full reward**: task-score feedback for mutation and reward-improvement acceptance.
- **No validator**: reward-blind mutation with every proposal admitted to validation-based selection.
- **Pointwise**: reward-blind mutation; parent and child outputs receive independent 1–10 scores, whose comparison gives a child win, parent win, or tie.
- **Symmetric pairwise**: reward-blind mutation; the judge compares both output orders. A consistent winner is retained; order disagreement becomes a tie.

Both judged configurations accept when child wins are at least parent wins across the three examples. All four use the same validation-based Pareto parent selector (Agrawal et al., 2025) and return the earliest maximum-validation candidate. All 24 candidates were fixed before held-out evaluation; no test score selected a candidate.

Validator inputs and generation. Both comparator rules use two judge calls per example, input prefixes of at most 2,000 characters, and output prefixes of at most 3,000 characters. Pointwise presents one output per call, whereas pairwise presents both outputs in both orders. The common call allowance therefore does not imply equal input tokens. Task generation receives the full rendered input. The judge input limit clips all 150 HotpotQA training inputs and 18 of 150 HoVer training inputs; omitted context can limit verification of an answer.

Task and judge requests use output limits of 2,048 and 8,192 tokens, respectively, with thinking requested off. Only visible response content is evaluated. Training includes a small number of responses with returned reasoning despite the requested setting; Appendix F.1 reports these observations. Empty task answers and malformed HoVer labels remain scored outcomes. Transport failures

permit at most three identical request attempts; invalid or truncated judge verdicts prevent completion of a run rather than becoming semantic ties.

C.2 Endpoint Scores under Shared Selection

Table 8: **Acceptance rules under shared validation-based selection.** Task scores and accepted proposals (%), mean±SD over three seeds. Parent and final selection use validation scores in every configuration. A dash denotes an unreported full-reward acceptance rate.

	HotpotQA		HoVer	
Configuration	**Task score**	**Accepted**	**Task score**	**Accepted**
Full reward	81.9±5.7	—	76.7±4.6	—
No validator	84.8±4.7	100.0	77.3±3.1	100.0
Pointwise	83.1±1.0	89.8±1.4	78.7±2.3	77.1±2.0
Symmetric pairwise	83.0±4.7	99.6±0.4	76.7±6.4	99.3±0.6

Table 8 pairs endpoint scores with acceptance rates. Symmetric pairwise and no validator have similar observed task scores, and both admit almost every proposal. This configuration therefore does not establish an endpoint benefit from the pairwise acceptance step; its tie policy, analyzed below, largely leaves selection to the downstream validation rules.

Table 9 reports every selected endpoint. All methods and seeds share the same 50 held-out questions per task, yielding 1,200 answers overall. Reported standard deviations describe three evolution seeds, not 150 distinct questions per configuration.

Table 9: **Controlled evolution results by seed.** Each entry averages the same 50 held-out questions for its task. All selected candidates use the common validation rule.

Task	**Seed**	**Full reward**	**No validator**	**Pointwise**	**Symmetric pairwise**
HotpotQA	42	81.67	83.95	84.09	83.97
HotpotQA	50	76.34	89.87	83.11	87.23
HotpotQA	51	87.78	80.48	82.18	77.87
HoVer	42	82.00	80.00	76.00	72.00
HoVer	50	74.00	74.00	80.00	84.00
HoVer	51	74.00	78.00	80.00	74.00

Paired symmetric-pairwise-minus-pointwise differences are -0.1 ± 4.2 points on HotpotQA and -2.0 ± 5.3 on HoVer. Differences from no validator are -1.7 ± 1.5 and -0.7 ± 9.5 points (mean±sample SD over seeds).

C.3 Acceptance and Tie Handling

Table 10: **How judgments become acceptance decisions.** Percentages, mean±SD over three seeds. Accepted and all-tie proposals use proposal denominators; tied comparisons use example denominators.

Task	**Comparator**	**Accepted proposals**	**All-tie proposals**	**Tied comparisons**
HotpotQA	Pointwise	89.8±1.4	72.2±2.9	89.8±1.0
HotpotQA	Symmetric pairwise	99.6±0.4	99.0±1.1	99.7±0.4
HoVer	Pointwise	77.1±2.0	46.6±1.7	77.5±1.3
HoVer	Symmetric pairwise	99.3±0.6	98.8±1.2	99.6±0.4

Symmetric pairwise accepts 1,046 of 1,050 HotpotQA proposals and 1,117 of 1,125 HoVer proposals; pointwise accepts 943 and 867, respectively, under the same aggregate tie policy and downstream selectors.

Two mechanisms produce pairwise ties. On HotpotQA, reversing the order changes the normalized decision in 42.0% of comparisons, while both orders explicitly tie in 57.7%. On HoVer, these rates are 72.3% and 27.3%. Reconciliation converts disagreement to a tie, and the acceptance rule then admits aggregate ties. Their combination preserves almost every proposal. This is consistent with thinking-off DeepSeek's strong position preference in Table 2b, measured on different answer pairs.

The primary configurations accept 51.9% (Val) and 51.6% (Elo) with Qwen3-8B, and 58.1% and 57.3% with gpt-5.4-mini (Appendix B.1). Their agents and validators differ, so this is not a matched no-validator control for the primary study, and stronger filtering alone does not establish better filtering. The design lesson is to evaluate comparison outcomes and tie policy jointly: reconciling answer order is useful only if the resulting decisions retain a meaningful acceptance criterion.

Held-out output quality. All 1,200 answers have nonempty visible content, and none returns a separate reasoning field. Table 11 reports output-limit stops, invalid labels, and zero-score answers by task and configuration. The 35 output-limit stops and 13 invalid HoVer labels remain scored outcomes; these categories may overlap.

Table 11: **Held-out answer outcomes.** Counts out of 150 responses per task/configuration (50 questions across three seeds). Categories overlap; output-limit stops and invalid labels remain in the score denominator.

Task	Configuration	Output limit reached	Invalid HoVer label	Zero score
HotpotQA	Full reward	3	—	11
HotpotQA	No validator	5	—	5
HotpotQA	Pointwise	7	—	5
HotpotQA	Symmetric pairwise	9	—	3
HoVer	Full reward	5	5	35
HoVer	No validator	2	2	34
HoVer	Pointwise	2	3	32
HoVer	Symmetric pairwise	2	3	35

D Validator Diagnostics

We compare DeepSeek V4 Flash and Qwen3.8-27B, both with reasoning enabled, on a frozen bank of HotpotQA and HoVer output pairs. The bank contains a designated control output and a constructed alternative at clear, moderate, and near difficulty levels. Each pair is judged in both presentation orders twice; each distinct output receives three independent pointwise scores. The diagnostic includes 5,480 pairwise and 8,091 pointwise calls per judge.

Fixed outputs hold candidate quality constant across validators. Agreement with the intended control tests error recognition; the remaining diagnostics measure repeatability and presentation sensitivity.

Synthetic-pair generation. Claude Opus 5 (recorded model identifier: `claude-opus-5`) authors pairs from source questions, reference answers, available solutions, and evaluator answer-extraction rules. Each batch contains approximately three to five questions and requests one control–alternative pair per question at each tier. The alternative should contain one annotated flaw. Clear, moderate, and near denote a surface-visible failure, an invalid intermediate inference, and a subtle error requiring verification or recomputation, respectively. These tiers describe intended flaw visibility, not evaluator-score gaps: an invalid argument can reach the correct answer. Reference access is used to construct the diagnostic, not supplied to the validators whose decisions are measured.

Example: answering the wrong relationship

HotpotQA question, abridged. Iola lies in a region of Kansas bounded by Woodson County in the northwest. Which county bounds it in the northeast?

Context summary. Southeast Kansas is bounded by Bourbon County in the northeast; Iola itself is in Allen County.

Control excerpt. "The question supplies the northwest corner and asks for the northeast one, so the region description supplies it directly. The answer is Bourbon County."

Alternative excerpt. "Iola is described as a city along the Neosho River in the northwestern part of Allen County, and it is the county seat there. [. . .] The answer is Allen County."

Intended flaw: clear tier. The alternative identifies Iola's county instead of the region's northeastern boundary. Both responses mention relevant geography, so the validator must check which relationship answers the question. Answer-token F1 is 1.0 for the control and 0.5 for the alternative.

Construction consistency checks. The pipeline verifies task/question identifiers and question prefixes, rejects duplicate questions and malformed records, and requires nonempty responses with recognized tiers. Evaluator checks remove ten HotpotQA pairs with control F1 below 0.8; all HoVer controls pass the label metric. A 12,000-character response-length filter removes no further pairs. These are structural and evaluator checks, not independent semantic verification of the annotated flaws.

The retained seed-42 set contains 1,370 pairs: 680 from 227 HotpotQA questions and 690 from 230 HoVer questions, with 457 clear, 456 moderate, and 457 near pairs. Controls are reused across tiers on 32 questions. Evaluator scores favor the control in 998 pairs and tie in 372, including 334 near pairs. Reported agreement is with the designated control, including on metric ties.

Position and length controls. Authoring assignments balance which response should be longer by question and tier; the build measures compliance and realized length balance. The adapter randomizes initial placement, then judges both orders twice. Table 13 reports selection of the longer output among decided calls on near pairs, excluding invalid calls and equal-length outputs. Unlike the separate Direct probe in Appendix D.3, it does not condition on stable winners. Length and content quality can covary, so this measures an association rather than a causal length effect.

Validator inputs. The runner caps questions at 2,000 characters and each response at 3,000, truncating 226 of 227 HotpotQA inputs but no responses. HoVer supplies claims and document-title/sentence-index pointers, not evidence passages. Judges may therefore lack context needed to verify subtle flaws.

Decision counts and position bias. Table 12 supplies the denominators for Table 2(a). Both methods use common valid pair–repeat records (repeat indices 0 and 1); an invalid call or missing score excludes the record from both. Pairwise order disagreement becomes a tie, and both methods receive half credit for ties. First-position selection instead pools decided raw calls before reconciliation, excluding ties and invalid calls. Its balance point is 50%, although aggregate balance can conceal order-sensitive individual decisions.

The near-pair length diagnostic has the same decided-call counts as the near-tier rows of Table 12.

Uncertainty estimates. We draw 5,000 question-level bootstrap resamples within each task and tier, keeping each question's pairs, orders, repeats, and scoring methods together. Each resample recomputes the pooled rates and paired difference under the same validity rules; their resampled standard deviation is the reported standard error. This accounts for correlated records within questions, not independent bank construction or evolution-run variation. Table 2b/c instead reports standard deviations across three independently authored banks.

D.1 Repeatability and Order Consistency

Table 13 reports order consistency, repeated-score stability, and position/length selection for the two thinking-enabled validators. Order consistency measures whether reversing the near-pair presentation

Table 12: **Decision counts and position bias on synthetic pairs.** Valid and invalid columns count common pair–repeat records. First-position selection uses decided raw calls; 50% is balanced, with either extreme indicating greater position bias.

Judge	Task	Tier	Valid n	Invalid	First-position selection (%)	Decided calls
DeepSeek	HotpotQA	Clear	446	8	49.9	895
	HoVer	Clear	450	10	53.2	901
	HotpotQA	Moderate	439	13	51.1	883
	HoVer	Moderate	442	18	54.8	889
	HotpotQA	Near	443	11	54.8	787
	HoVer	Near	437	23	57.8	784
Qwen	HotpotQA	Clear	438	16	50.0	908
	HoVer	Clear	436	24	52.6	920
	HotpotQA	Moderate	431	21	50.3	904
	HoVer	Moderate	431	29	53.8	920
	HotpotQA	Near	438	16	46.9	908
	HoVer	Near	426	34	53.7	918

preserves the winner; score stability measures exact agreement across repeated pointwise scores. These supplement control agreement: repeatable or order-consistent decisions can still be wrong.

Table 13: **Order consistency, repeatability, and length preference (%).** Both validators use thinking. Order consistency and selection rates concern near-tier pairs; score stability measures exact agreement across repeated pointwise scores. Selection rates report estimate±SE.

Validator	Task	Order consistency	Score stability	First position	Longer output
DeepSeek V4 Flash	HotpotQA	74.4	42.5	54.8±1.1	63.0±3.0
	HoVer	65.1	30.5	57.8±1.4	70.8±2.5
Qwen3.8-27B	HotpotQA	85.5	45.7	46.9±1.1	51.5±2.9
	HoVer	72.8	35.3	53.7±1.4	58.3±2.6

DeepSeek has 63 unparsed pairwise calls out of 5,480 and 31 missing pointwise scores out of 8,091; Qwen has 1 and 209, respectively. Neither has a failed pairwise request. The runner uses whole-text parsing and can read reasoning text when visible content is empty. Table 2a uses common valid records and the shared tie rule specified above.

D.2 Thinking, Agreement, and Presentation Bias Across Five Models

Table 14 reports all five validators in both thinking modes, pooled and separately for each of the six tasks. The banks use seeds $\{42, 50, 51\}$ and cover HotpotQA, HoVer, IFBench, PUPA, the American Invitational Mathematics Examination (AIME), and LiveBench-Math (White et al., 2024) at all three tiers. Models and modes share each bank, with two orders and two repeats per pair. Each configuration has 14,152, 14,144, and 14,080 scheduled pairwise calls in the three banks, respectively. This study is separate from the two-validator HotpotQA/HoVer comparison above; in particular, its Qwen3.6-27B is not the Qwen3.8-27B in Table 2a.

Within each bank, first-position selection pools decided raw calls across all tiers, excluding ties and invalid calls. Longer-output selection uses the same calls but additionally excludes equal-length outputs. The pooled rows combine task counts before calculating percentages, so tasks contribute in proportion to their eligible-call counts. The table reports mean and sample SD across three bank-level percentages, not variation over evolution runs. Its pooled position and length columns reproduce Table 2b/c.

Clear-control agreement uses decided clear-tier calls, excluding ties rather than assigning half credit as in Table 2a. With thinking, pooled clear agreement improves for four validators and changes little for Gemma; pooled position balance improves for all five, while length selection varies by model.

Task-level exceptions locate where additional validation is needed: PUPA clear-control agreement remains near or below chance in both modes, and Gemma's IFBench position preference moves farther from balance with thinking. Aggregate improvements therefore need checking on the intended task.

Thinking-off and thinking-on configurations use output limits of 1,024 and 16,384 tokens, respectively. The comparison therefore evaluates mode and generation allowance together, not reasoning at equal compute; complete serving settings are unavailable.

Table 14: **Thinking-mode diagnostics across all six tasks.** Percentages, mean±SD across three authored banks. Off/on denotes thinking. Clear agreement measures selection of the control among decided clear-tier calls; position and length rates pool all tiers. For selection rates, 50% denotes balance, not accuracy.

Task	Clear agreement ↑		First-position selection		Longer-output selection	
	Off	On	Off	On	Off	On
Qwen3-8B						
All tasks	55.3±0.4	77.6±0.9	94.3±0.2	52.2±0.2	55.3±0.2	70.9±0.4
HotpotQA	56.1±0.9	84.5±1.3	94.2±0.4	52.2±0.9	54.5±0.8	64.9±0.7
HoVer	61.5±1.3	73.1±1.3	89.2±0.6	52.1±0.7	60.1±0.8	74.5±0.5
IFBench	53.4±2.1	66.3±6.8	95.6±0.3	57.3±1.0	54.4±0.3	85.7±0.3
PUPA	51.6±0.5	50.9±4.4	93.8±1.5	50.9±1.4	56.2±1.5	87.1±0.6
AIME	51.9±0.2	98.7±0.7	98.4±0.3	49.6±1.2	51.6±0.3	56.1±2.1
LiveBench-Math	54.2±0.8	93.1±0.5	96.6±0.4	50.1±0.9	53.2±0.4	56.0±1.8
Gemma-4-26B						
All tasks	78.2±0.5	77.7±0.7	44.8±0.2	53.7±0.1	69.8±0.4	72.5±0.6
HotpotQA	92.3±0.6	92.6±0.6	43.8±0.6	54.6±0.3	58.0±1.8	62.7±2.0
HoVer	79.2±2.8	78.2±3.6	46.1±0.7	54.3±0.5	67.4±0.4	70.5±1.2
IFBench	71.4±1.6	71.4±2.6	45.5±0.9	60.7±1.6	74.3±1.1	77.4±0.3
PUPA	48.6±2.9	48.2±2.8	46.1±1.1	50.3±0.9	87.3±0.8	88.0±0.6
AIME	82.8±2.1	80.3±0.7	45.9±0.5	49.6±0.5	71.6±2.3	73.3±2.4
LiveBench-Math	86.5±1.8	86.6±1.4	42.4±0.5	50.2±1.0	66.8±2.1	68.7±2.0
Qwen3.6-27B						
All tasks	79.3±0.5	88.8±0.4	71.1±0.2	50.1±0.3	61.4±0.7	58.4±0.1
HotpotQA	90.3±0.9	97.7±0.3	67.2±0.5	48.9±0.3	54.1±1.1	53.3±0.3
HoVer	89.7±1.6	92.9±0.3	68.4±0.3	49.2±1.1	59.5±0.5	59.1±1.6
IFBench	74.3±0.9	97.9±0.4	76.6±0.4	51.9±0.4	64.5±0.1	62.9±2.6
PUPA	43.5±1.5	36.1±1.5	65.3±0.6	51.3±0.2	76.6±1.0	73.1±1.8
AIME	83.5±2.1	99.6±0.2	76.2±1.6	49.9±0.4	59.6±1.4	52.0±1.1
LiveBench-Math	84.9±0.6	98.0±0.4	74.3±0.2	49.7±0.2	57.9±2.5	51.7±1.9
Qwen3.5-122B						
All tasks	74.9±1.3	88.2±0.6	42.1±0.3	51.3±0.1	67.4±0.7	59.0±0.4
HotpotQA	86.2±0.7	96.1±0.5	39.6±0.7	51.1±0.6	58.5±2.1	54.7±1.7
HoVer	85.3±2.3	89.2±1.5	40.7±0.5	51.5±0.3	63.8±1.2	61.4±1.3
IFBench	63.8±4.5	98.2±0.5	37.2±0.8	53.3±0.5	82.2±0.5	60.2±1.9
PUPA	42.8±2.5	39.8±2.0	43.6±1.2	52.1±0.7	82.6±1.5	75.9±1.3
AIME	80.8±3.0	99.7±0.1	50.9±0.7	50.2±0.4	66.0±1.2	51.7±1.5
LiveBench-Math	80.5±2.0	97.3±0.8	43.7±1.5	49.6±0.4	59.1±1.5	51.8±1.8

Continued on next page

Table 14 (continued)

Task	Clear agreement ↑		First-position selection		Longer-output selection	
	Off	On	Off	On	Off	On
DeepSeek V4 Flash						
All tasks	54.0±0.2	87.8±0.3	96.2±0.2	53.0±0.2	54.1±0.3	61.5±0.9
HotpotQA	58.4±0.6	95.6±0.9	95.2±0.2	52.1±0.4	53.9±0.5	57.0±1.7
HoVer	51.9±0.5	89.4±2.4	99.1±0.1	54.8±0.7	51.7±0.2	61.1±2.4
IFBench	51.3±1.7	88.4±2.9	96.0±0.1	55.8±0.5	55.0±0.1	71.8±0.5
PUPA	52.0±0.5	43.8±0.9	91.7±1.3	54.1±0.4	58.6±1.2	81.2±1.1
AIME	53.2±0.9	99.9±0.1	98.5±0.3	50.0±0.4	52.5±0.5	51.6±1.6
LiveBench-Math	56.1±1.4	99.1±0.6	96.7±0.1	50.6±0.2	53.6±0.3	50.5±2.3

D.3 Bias on Evolved Output Pairs

This probe asks whether presentation sensitivity also appears on outputs produced during evolution, rather than only on authored flaws. We mine 60 near-tie pairs (example-level task-metric gap at most 0.1) and 60 clear-gap pairs (gap at least 0.5) from HotpotQA, HoVer, and IFBench checkpoints. PUPA is excluded because its evaluator is itself LLM-based. Every judge sees the same pairs twice with A/B order swapped, using the Direct prompt and temperature zero.

Table 15 separates order consistency, clear-gap accuracy, and length preference on these evolved outputs. No-thinking Qwen3-8B is strongly position-driven with near-chance clear-gap accuracy. Randomized placement avoids a fixed parent/child advantage but does not correct individual judgments; Appendix C examines how order reconciliation and tie acceptance interact.

Table 15: **Judge sensitivity under order reversal.** Rates are percentages. Clear-gap accuracy uses decided calls; length selection uses pairs with a consistent winner. Parentheses give the corresponding counts; a dash denotes an unavailable estimate.

Judge	Near-pair order consistency	Clear-gap accuracy	Longer output wins
Qwen3-8B (thinking on)	57	72 (81/113)	55 (38/69)
Qwen3-8B (thinking off; Direct)	2	54 (65/120)	—
Claude Haiku 4.5	37	62 (74/119)	52 (23/44)
Claude Sonnet 5	58	82 (50/61)	46 (13/28)

D.4 Pointwise Scores and Acceptance Behavior

Independent output scores introduce a scale-resolution question: can the judge assign different scores to changes small enough to matter during evolution? The earlier pointwise configuration uses a 1–10 integer scale; equal or unparseable scores become ties. These scores also select parents and final checkpoints, unlike the validation-based selection in Val. Table 16 describes the behavior of these complete configurations; Appendix C isolates acceptance comparisons under common downstream selectors.

Table 16: **Rejection and tie rates in the earlier Qwen3-8B configurations.** Percentages pooled over five seeds. Rejection uses proposal denominators; ties use the three example comparisons per proposal.

Task	Comparator	Proposals	Rejected proposals (%)	Tied comparisons (%)
HotpotQA	Direct	1750	48.4	0.2
	Pointwise	1750	26.8	53.7
IFBench	Direct	1000	48.5	0.3
	Pointwise	1000	22.1	79.7
HoVer	Direct	1875	49.4	0.0
	Pointwise	1875	36.5	60.8
PUPA	Direct	625	45.6	0.6
	Pointwise	625	29.6	65.5

Direct pairwise validation rejects 46–49% of proposals in this campaign, versus 22–37% for pointwise, whose example-level ties reach 54–80%. These are Direct diagnostics, not measurements of the primary Adaptive validator. An offline repeatability probe rescored 120 outputs three times: 115 received the same integer on all calls, $350/360$ repeated-score pairs matched exactly, and no preference sign flipped. Scores concentrated on five integers $(2, 6, 7, 8, 9)$. Repeatable scores can thus remain coarse, making the treatment of ties consequential.

Several Direct-evolved HoVer prompts used labels such as `INSUFFICIENTLY_SUPPORTED`, which the permissive evaluator credited as `SUPPORTED`. This observation comes from the earlier diagnostic experiment, but the primary evaluator shares that parsing rule. It illustrates why evaluator score and valid task behavior must be distinguished. The controlled study instead uses provided evidence and strict final-label parsing.

D.5 Changing the Validator Model

Table 17 holds the Qwen3-8B generator fixed while comparing self-validation with Haiku. Both use Adaptive judging and validation-based parent and final selection. Unlike the primary study, this configuration generates fresh parent/child outputs on three validation inputs per proposal. Their labels are not shown to the validator; changing the comparison inputs adds inference without revealing answers. Split construction also differs, so this study has its own full-reward reference. Haiku exceeds self-validation on all four observed scores, illustrating that the validator can be replaced while the generator and selector stay fixed.

Table 17: **Self-validation is not required.** Qwen3-8B test scores (%), one run per configuration. Both validator rows use Adaptive judging with validation-score parent selection, not Elo. Full reward is a shared reference; HoVer uses restricted inputs and permissive scoring.

Validator / reference	HotpotQA	IFBench	HoVer	PUPA
Full reward (reference)	77.0	78.3	50.0	62.1
Qwen3-8B (self-validator)	78.5	80.6	52.0	59.4
Haiku 4.5 (external validator)	80.5	82.2	56.0	61.0

E Program Evolution

Program evolution tests whether a local comparison can be inserted into engines whose candidates are executable code. Execution checks answer whether a program runs and passes specified tests; an LLM comparison supplies an additional judgment about the proposed change. Table 18 compares five configurations. Execution checks uses programmatic checks alone; Code judge receives child source without parent execution evidence. Pairwise + checks (ours) presents both programs, proxy outputs, and proxy scores after execution and regression checks. Full reward + judge uses source comparison

in ADRS and execution-assisted pairwise comparison in ShinkaEvolve. These configurations vary information and selection rules, not only reward access.

Circle packing shows that the pairwise interface can improve the reference in both engines, while scheduling and signal processing favor other feedback configurations. Portability of the interface does not require the same feedback rule to lead on every objective.

Scores use the recorded terminal evaluation, not the last-created program or the maximum observed score.

Circle-packing values are sums of radii for 26 circles; ADRS's stored target ratio is multiplied by 2.635. Scheduling reports $10^6/(1 + \text{makespan})$, not makespan itself, so larger is better. Signal processing uses its recorded evaluator score. Prime counting uses an accuracy–runtime objective, not an accuracy percentage. All five final evaluator outputs report correctness; differences in these runtime-sensitive scores do not establish a speed ranking from one evaluation per program.

Table 18: **Program evolution with an external validator.** Terminal scores, one run per configuration (seed 50), with Qwen3-8B generation and Haiku 4.5 validation. Larger is better within each task.

Engine	Task	Full reward	Code judge	Execution checks	Pairwise + checks (ours)	Full reward + judge
ADRS	Circle packing	1.76	2.25	1.84	2.16	1.88
	Signal processing	0.63	0.59	0.63	0.35	0.42
	Scheduling	3787.9	3636.4	3257.3	3174.6	3289.5
ShinkaEvolve	Circle packing	1.37	2.26	1.39	1.90	1.87
	Prime counting	99.63	99.74	99.73	99.74	99.75

F Evaluation Budgets and Inference Cost

Reward access and computational cost are different quantities: replacing an evaluator's feedback does not necessarily avoid its execution, and LLM comparisons add inference. Table 19 reports proposal limits and evaluation sizes; the following subsection reports observed calls where available. Physical-call and GPU-hour traces are incomplete outside the controlled study, so configured limits do not establish matched compute.

Table 19: **Task budgets and evaluation sizes.** Configured iteration limits, validation sizes $|\mathcal{V}|$, and test sizes. Program tasks use executable evaluators rather than prompt-task data splits. Iteration limits do not imply equal inference cost.

Task	Domain	Engine	Max iter	$\lvert\mathcal{V}\rvert$	Test size
HotpotQA	Reasoning	GEPA	350	30	50
IFBench	Instructions	GEPA	200	30	30
HoVer	Reasoning	GEPA	375	30	50
PUPA	Rewriting	GEPA	125	30	30
Scheduling	Programs	ADRS	500	*Not applicable*	*Not applicable*
Circle packing	Programs	ADRS	500	*Not applicable*	*Not applicable*
Signal processing	Programs	ADRS	500	*Not applicable*	*Not applicable*
Circle packing	Programs	ShinkaEvolve	400	*Not applicable*	*Not applicable*
Prime counting	Programs	ShinkaEvolve	400	*Not applicable*	*Not applicable*

F.1 Observed Calls in the Controlled Study

The controlled experiment records physical requests as well as completed logical calls. Table 20 aggregates the full three-seed campaign by model role. Task-model calls include reflection, minibatch

generation, and validation, so their count is not an example denominator. Shared serving logs support role totals rather than a reliable allocation of all recovery costs to individual configurations.

Table 20: **Observed inference in the controlled experiment.** Extra attempts are transport retries. Token counts sum returned usage fields; usage for failed attempts is unavailable, so training totals are observed lower bounds.

Phase and role	Logical calls	Extra attempts	Observed tokens
Training: task model	267,680	28	783,715,192
Training: validator	26,100	2	32,242,713
Held-out: task model	1,200	0	3,202,215

Across training task-model calls, 76 outputs have empty visible content and 9,684 nonempty outputs reach the generation limit. Seventeen responses return a separate reasoning field despite thinking being requested off. Validator responses have no empty outputs, output-limit stops, or returned reasoning. All 30 extra physical attempts recover without changing the request body. The held-out stage needs no retries and has complete token usage records.

Reward substitution changes which information guides evolution; pairwise judging itself adds inference. The reported token counts describe this campaign and do not establish that one reward regime is cheaper than another.

G EXAMPLES OF EVOLVED ARTIFACTS

The following prompt and program excerpts illustrate how evolution changes the artifact supplied to the host engine. They are examples rather than additional performance evaluations.

G.1 AN EVOLVED QUESTION-ANSWERING PROMPT

The first example is the validation-selected prompt from the controlled symmetric-pairwise HotpotQA run with seed 42. It specifies intermediate operations for resolving multi-hop questions: identify anchor entities, follow their relationships, and check all question constraints. Its final-answer instruction separates the answer from the reasoning. The excerpts below are reproduced verbatim; they illustrate the resulting instructions rather than isolate which instruction causes the score.

HotpotQA: resolving entities and formatting the answer

```
   * **Chain Resolution**: If the question involves a chain of
   ↪  entities (e.g., "Who is the brother of X who played in Y?"),
   ↪  resolve one step at a time.
     1. Find information about X.
     2. Find the entity related to X (e.g., the brother).
     3. Use that new entity as the anchor to find the final answer
     ↪  (e.g., the team they played for).


5. **Final Output**:
   * Provide a concise, direct final answer.
   * If the answer is a name, provide just the name(s).
   * If the answer is a value, include the unit if provided in the
   ↪  text (e.g., "575 acres").
   * Do not include extraneous information, reasoning, or "Yes/No"
   ↪  prefixes in the Final Answer line unless the question
   ↪  specifically requires it.
```

G.2 AN EVOLVED CIRCLE-PACKING PROGRAM

The second example comes from ShinkaEvolve with Pairwise + checks, using Qwen3-8B generation and Haiku judging. The saved program itself searches over circle-center configurations. Its update recombines and perturbs centers, clips them to the square, and recomputes radii before adding the configuration to its population. This internal search is part of the generated program, distinct from ShinkaEvolve's outer archive of programs. The excerpt illustrates executable task logic produced through the shared acceptance interface.

Circle packing: updating centers and radii

```
    child_centers = crossover(parent1[0], parent2[0])

    # Mutation
    child_centers = mutate(child_centers, mutation_rate)

    # Clip to ensure everything is inside the unit square
    child_centers = np.clip(child_centers, 0.01, 0.99)

    # Compute radii for child
    child_radii = compute_max_radii(child_centers)
    next_generation.append((child_centers, child_radii,
    ↪  np.sum(child_radii)))
```