JET: JUDGE-GUIDED EVOLUTION AT TEST TIME FOR AGENT PROGRAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

An agent's executable program governs how it uses tools, processes observations, and responds to failures. Evolving this program at test time can help adaptation, but deciding which changes to retain is difficult when true rewards are unavailable. Execution traces provide evidence of agent behavior, yet interpreting that evidence requires a judge that remains useful as tasks and candidate programs change. We introduce Judge-Guided Evolution at Test Time (JET), which evolves an executable judge on labeled source trajectories, then freezes and transfers it to guide target-side program evolution. The judge supplies scores and diagnostic feedback without target evaluator access or model-weight updates. On unseen WebShop tasks, JET achieves approximately 13% higher mean reward than fixed-rubric guidance when evolution begins from an unevolved program (cold start) and 4% higher when it begins from one already optimized on source tasks (warm start), with a 36% relative improvement in cold-start exact success. An exact-judge control on PushT, where the judge reconstructs the scoring rule from observations, shows that without judge error, program search becomes the bottleneck. Analyses identify useful reward-prediction logic in the evolved code and show that better final selection alone cannot explain the gains. These results support executable judge transfer for program adaptation under evaluator-preserving task shifts.

1 INTRODUCTION

Building a useful language-model agent requires decisions about the software around the model: which information to retain, which tools to invoke, and how to respond when an attempt fails (Yao et al., 2022b; Shinn et al., 2023). These decisions are implemented in the agent's *harness*, an executable program that organizes its interaction with a task environment. We use *agent program* to cover both LLM harnesses and executable control policies. Revising this program provides a way to change the agent's behavior while keeping the underlying model fixed. *Harness evolution* automates this process by proposing code changes, executing candidate programs, and using feedback to guide further edits (Lee et al., 2026). It fits the broader perspective of *heuristic learning*, in which executable procedures improve through edits informed by rewards, tests, and execution records (Weng, 2026).

Such a search relies on evaluation to decide which changes to retain and what to revise next. Task rewards provide a direct measure of candidate performance, but at test time an agent may only be able to execute programs and inspect their behavior without access to these rewards. A judge must then infer task completion from observable behavior and provide useful feedback about remaining errors (Zhuge et al., 2025). If it systematically undervalues partial progress or misidentifies a failure, its scores and diagnostics can direct search away from useful changes. This motivates our central research question:

> ***Can a judge developed on source trajectories with known rewards guide program evolution on target tasks whose evaluator outputs are withheld?***

Criteria for assessing behavior may remain useful even when the actions needed for success change. Prior reward-model work provides evidence for this possibility (Xia et al., 2025; Fu et al., 2017; Gleave et al., 2020). Our focus is an executable judge whose evidence extraction, LLM assessment,

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.

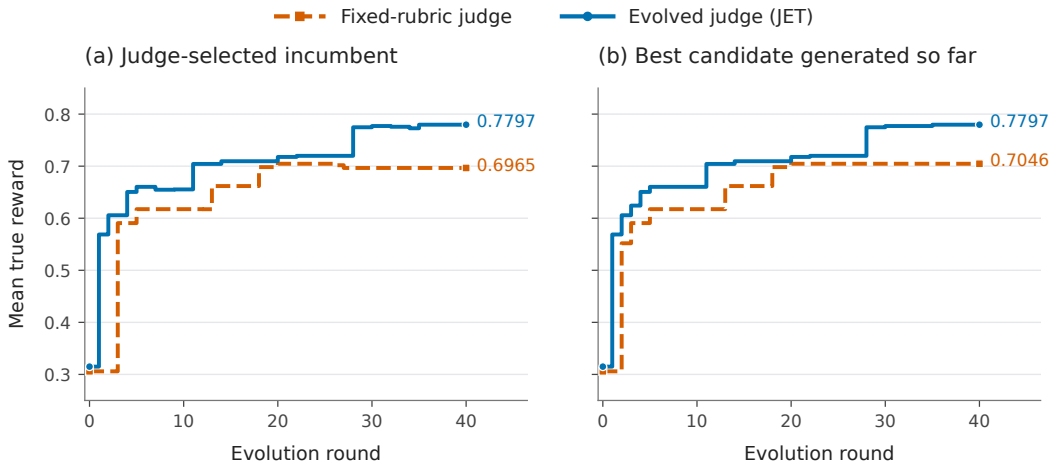


Figure 1: **Judge-guided evolution on WEBSHOP.** Mean true reward across five independent cold-start campaigns at each round for (a) the incumbent and (b) the best candidate generated so far within each campaign. Rewards are audited post hoc; Appendix D details the aggregation.

and score adjustment can themselves be revised through program evolution. The challenge is twofold: the judge must transfer across the source–target shift and remain useful as program search changes the behaviors it evaluates (Gao et al., 2023). We evaluate this transfer through both reward prediction and the target-side program improvements it supports.

We introduce JET, a two-stage procedure that first evolves an executable judge on labeled source trajectories and selects it using a separate source validation split. The judge is then frozen and supplies scores and diagnostics for target-side program evolution. Underlying model weights remain fixed, and target evaluator outputs are used only for post-hoc evaluation.

WEBSHOP supplies the main test of judge transfer: a judge developed on clothing trajectories guides adaptation to non-clothing shopping tasks, and the selected programs are also evaluated on previously unseen target sessions. Figure 1 previews the primary cold-start searches: evolved-judge guidance reaches higher observed mean reward for both the selected program and the best candidate generated so far. PUSHT serves as an exact-judge control: the judge reconstructs the unchanged geometric evaluation rule from observations, which isolates program search from judge error. Both benchmarks preserve the evaluation rule across the source–target shift; they examine adaptation when tasks or dynamics change while the scoring criterion remains fixed.

Our contributions are:

- **Executable judge evolution for test-time adaptation.** We introduce JET, which optimizes an executable judge's evidence extraction, LLM assessment, and score adjustment on labeled source trajectories. The frozen judge then supplies scores and diagnostics for test-time agent-program evolution, without target evaluator access or updates to underlying model weights.

- **Evaluation under evaluator-preserving shifts.** On unseen WEBSHOP tasks, JET improves mean reward over fixed-rubric guidance by approximately 13% when starting from an unevolved program and 4% when starting from a program optimized on source tasks. From the unevolved initialization, exact success improves by 36% relative to fixed-rubric guidance.

- **Analysis of how evaluation shapes program search.** Source-holdout controls identify executable reward-prediction rules. Primary campaign results show that better final selection alone cannot explain the mean performance gap between guidance conditions. Complementary online comparisons examine executable evolution beyond prompt optimization and the roles of textual diagnostics and numerical score adjustments.

2 Related Work

Program optimization and heuristic learning. DSPy and Promptbreeder optimize language-model pipelines and prompts (Khattab et al., 2023; Fernando et al., 2024); ADAS, STOP, and AFlow search executable agents, optimizer code, or workflows (Hu et al., 2024; Zelikman et al., 2024; Zhang et al., 2025a). AlphaEvolve and GEPA further illustrate code- and prompt-based improvement from evaluation feedback (Novikov et al., 2025; Agrawal et al., 2025). The heuristic-learning perspective describes a broader process of maintaining software through code edits informed by rewards, tests, logs, and replays (Weng, 2026). We study the evaluation component of this process: whether source supervision can produce an executable judge that remains useful when the target evaluator is inaccessible.

Harness generation. Meta-Harness searches harness code using prior programs, execution traces, and task-specific scores on its search tasks (Lee et al., 2026). JIT-Agent trains a model to generate and repair task-conditioned harnesses (Zhang et al., 2026). Concurrent TTHE work studies test-time harness evolution using unlabeled execution feedback (Nie et al., 2026). We focus on source-supervised evolution of the evaluation procedure itself: executable judge code is optimized against known source outcomes, then frozen and transferred to guide test-time program search.

Learned judges and reward transfer. LLM judges and generative verifiers assess outputs or reasoning trajectories (Zheng et al., 2023; Zhang et al., 2025b), while Similar learns step-wise, multidimensional signals for interactive agents (Miao et al., 2025). AgentRM explicitly studies reward-model transfer and uses learned rewards for best-of-N and beam search on agent tasks (Xia et al., 2025). Earlier work also learns reward models that transfer across changed initial-state distributions (Reddy et al., 2020). These works establish evaluation transfer as a useful direction. Our focus is evolving the judge as executable software and using it to guide persistent program rewrites across tasks. The judge combines evidence extraction, LLM assessment, and score adjustment, and returns textual diagnostics, while its underlying model weights remain fixed.

3 Method: Evolving and transferring the judge

3.1 Problem Setting

We consider a task space $\mathcal{T}$ and a space $\mathcal{X}$ of trajectories observable during ordinary agent execution. An agent program is an executable program π that maps a task $\tau \in \mathcal{T}$ to an observable trajectory $x = \pi(\tau) \in \mathcal{X}$. An evaluator maps a task–trajectory pair to a scalar reward $R(\tau, x) \in \mathbb{R}$, which we call the true reward. Evaluator outputs are available for labeled source trajectories used to develop the judge, but are withheld on target tasks during adaptation. We include the deployment-visible task specification in x, so the judge can assess behavior against the requested goal.

A *proposer* is a code-generating model that produces candidate rewrites from the current program and available feedback. Selecting among these candidates requires an evaluation signal. Directly evaluated search uses R, whereas we study test-time search without querying the target environment evaluator. To guide search under this restriction, we develop a judge on labeled source tasks $\mathcal{T}_{\mathrm{src}}$, where evaluator outputs are available, and transfer it to reward-hidden target tasks $\mathcal{T}_{\mathrm{adapt}} = \{\tau_i\}_{i=1}^{n}$. Here, *reward-hidden* means that evaluator outputs are withheld; the evaluation rule need not be unknown or unreconstructible. During target adaptation, the proposer, transferred judge, and selection rule may access target-task instructions and deployment-visible trajectories. Their access excludes target evaluator outputs and derived fields, which remain outside the adaptation workspace and are used only for post-hoc evaluation.

Our objective is to maximize mean true reward on $\mathcal{T}_{\mathrm{adapt}}$. Because target trajectories inform program search, performance on this set measures adaptation to the observed tasks rather than generalization to unseen tasks (Wang et al., 2026). Candidate checks enforce the permitted observation and execution interfaces. The procedure requires executable candidates, serializable trajectories, source evaluator access, and repeatable target execution; implementation details appear in Appendix A.

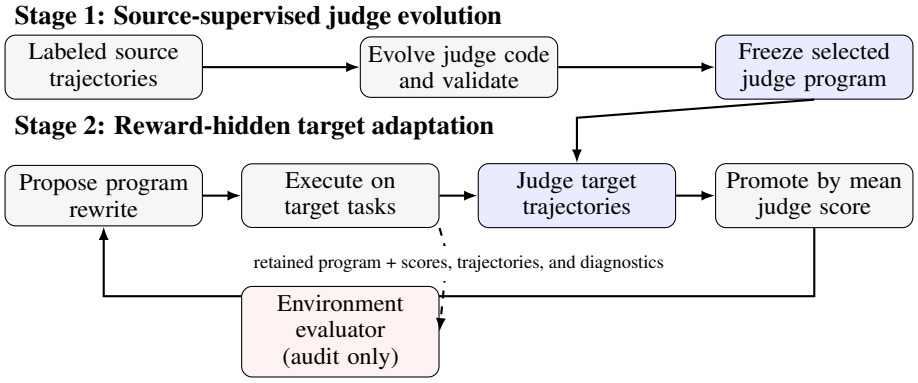


Figure 2: Two-stage evaluator transfer. Source labels guide judge-program development. The frozen judge then supplies scores for promotion and diagnostics for subsequent program rewrites. Feedback from rejected candidates is retained. Environment-provided target evaluator outputs are isolated for auditing and have no return path to adaptation.

3.2 Stage 1: Evolve an Executable Judge

A judge J maps an observable trajectory $x \in \mathcal{X}$ to a scalar estimate $\hat{r} = J_{\text{score}}(x)$ and may additionally produce structured diagnostics $J_{\text{diag}}(x)$ for the proposer. We represent J as an executable program whose prompts, trajectory preprocessing, deterministic computations, calibration logic, and output parsing may be revised during source-side evolution. The weights of any underlying inference models remain fixed.

Given labeled source trajectories $\mathcal{D} = \{(x_i, r_i)\}$, judge evolution minimizes the mean absolute error between predicted scores $J_{\text{score}}(x_i)$ and source evaluator rewards r_i. Source optimization supervises scalar reward prediction while diagnostic usefulness is examined through the downstream feedback interventions in Section 5.4.

At each round, a proposer receives the current judge code and source-task feedback and emits a candidate judge program. Training loss determines the next evolutionary parent and loss on a private validation split selects the final candidate after a fixed search budget. A source holdout excluded from judge development is used for post-selection reporting. The selected judge is then frozen: its code, parameters, and calibration logic cannot change during target adaptation. Any underlying LLM continues to operate with the frozen judge's inference procedure.

3.3 Stage 2: Freeze the Judge and Evolve the Agent Program

Let $\pi_t^\star$ denote the highest-scoring eligible program observed through round t. At each round, the proposer receives the code and judge score of the current best program $\pi_{t-1}^\star$ together with the deployment-visible trajectories and frozen-judge feedback from the most recently evaluated candidates. The latter feedback remains available even when those candidates were rejected, so the current best program and the proposer's feedback state need not describe the same program. Using this feedback, the proposer generates a set of candidate rewrites $\mathcal{C}_t$. Each candidate is executed on all target tasks and scored by

$$S_J(\pi) = \frac{1}{n} \sum_{i=1}^{n} J_{\text{score}}(\pi(\tau_i)) \,. \tag{1}$$

Let $\mathcal{C}_t^{\text{elig}} \subseteq \mathcal{C}_t$ contain candidates satisfying all interface and, when enabled, source constraints; when this set is nonempty, let $\bar{\pi}_t$ be its highest-scoring member. The current best program is updated only when $\bar{\pi}_t$ has a higher judge score:

$$\pi_t^\star = \begin{cases} \bar{\pi}_t, & \mathcal{C}_t^{\text{elig}} \neq \varnothing \text{ and } S_J(\bar{\pi}_t) > S_J(\pi_{t-1}^\star), \\ \pi_{t-1}^\star, & \text{otherwise.} \end{cases} \tag{2}$$

Selection in the proposed reward-hidden conditions uses no target evaluator output. The initial program may be a cold seed program or a warm program previously optimized using labeled source

tasks. The transferred judge supplies scores and diagnostics: its scores determine promotion and enter proposer feedback, while its diagnostics and the trajectories of both accepted and rejected candidates also inform subsequent rewrites.

3.4 From Reward Prediction to Search Guidance

The judge learns to assess realized behavior, while target-side search discovers how to produce it. Its average prediction error is only one measure of usefulness (Frick et al., 2025; Lou et al., 2025): promotion depends on the ordering of candidate mean scores in Equation 1. For two candidates with different true mean rewards, a useful judge assigns the higher score to the better candidate. Agreement near promotion decisions is therefore relevant even when overall prediction error is small. The core procedure requires no source-evaluator queries during target adaptation. An optional source constraint can restrict eligible candidates to those meeting an evaluator-measured floor on a fixed source check set (Appendix B).

4 Experimental Setup

4.1 Tasks and Transfer Settings

WebShop is used to test interactive shopping under a clothing-to-non-clothing shift (Yao et al., 2022a). Clothing source tasks are divided into training, validation, and holdout sets for judge development, selection, and evaluation. Adaptation uses non-clothing tasks and all splits are session-disjoint. Additionally, held-out tasks never guide adaptation.

PushT tests continuous control by pushing a T-shaped block into a goal (Chi et al., 2025). Source tasks begin near the goal without damping; target tasks have larger initial displacement and nonzero damping. The geometric evaluation rule is unchanged and exactly reconstructible from visible poses, making this an exact-judge control. Dataset construction and execution settings are collected in Appendix A.

4.2 Comparison Conditions

We compare no target evolution, proposer self-selection, and guidance from the fixed-rubric or evolved judge. The fixed-rubric judge uses a predefined task-specific rubric and scoring procedure, without source-label optimization. For WebShop proposer self-selection, the proposer assigns each candidate a scalar self-score after reading its reward-scrubbed trajectories, with the candidate with the highest self-score being selected. The evolved judge is developed on source trajectories and selected by source validation MAE, and both judges remain frozen during target adaptation. In the primary WebShop experiments, the seed judge that initializes evolution is distinct from the fixed-rubric judge. A *warm start* uses a source-optimized program; a *cold start* uses the unevolved seed. Each primary condition is run as five independent campaigns; a campaign is one complete 40-round evolution run from a given initialization. We report campaign means and standard deviations. Within each comparison and initialization, adaptive conditions use the same task sets and test-time proposal budgets. Under full judge guidance, mean scalar estimates determine promotion, while scores and textual diagnostics inform rewrites. Source-constrained conditions, labeled *source constraint* in Table 1, additionally query labeled source tasks to enforce a fixed performance floor (Appendix B).

4.3 Evaluation Metrics

Primary program metrics are mean true reward in WebShop and mean final goal coverage in PushT, the fraction of goal area overlapped by the block. Judge fidelity is measured by MAE and correlation with true rewards. Environment-provided target evaluator outputs are withheld during adaptation. In PushT, the frozen judge nevertheless reconstructs the final-coverage metric exactly from visible poses. For each run, *oracle-best* is the highest true performance among its generated candidates; *selection regret* is the gap between that performance and the selected candidate's audited performance. These diagnostics are computed within each campaign and then averaged; source-constrained accounting is given in Appendix B.

Table 1: Test-time evolution across five independent campaigns. Panel A: WEBSHOP reward; Panel B: PUSHT coverage (exact-judge control). Selected values are mean ± SD; other columns are means. Bold marks the best selected mean and gain per benchmark and initialization; grey denotes JET. Environment evaluator outputs are withheld online; the PUSHT judge reconstructs coverage exactly.

Start	Selection signal	Initial	Selected	Δ	Regret ↓
Panel A: WEBSHOP					
Warm	None	0.6256	0.6256 ± 0.0175	–	–
Warm	Fixed-rubric judge	0.6256	0.6589 ± 0.0299	+0.0333	0.0198
Warm	Evolved judge	0.6256	0.6951 ± 0.0706	+0.0694	0.0000
Warm	Evolved + source constraint	0.6250	$\mathbf{0.7034} \pm 0.0122$	**+0.0784**	0.0111
Cold	Self-selection	0.3043	0.6137 ± 0.0699	+0.3094	0.0121
Cold	Fixed-rubric judge	0.3059	0.6965 ± 0.0268	+0.3906	0.0081
Cold	Evolved judge	0.3150	$\mathbf{0.7797} \pm 0.0702$	**+0.4647**	0.0000
Cold	Evolved + source constraint	0.3158	0.7100 ± 0.0109	+0.3942	0.0000
Panel B: PUSHT					
Warm	None	0.3709	0.3709 ± 0.0008	–	–
Warm	Fixed-rubric judge	0.3709	0.6033 ± 0.0087	+0.2324	0.0475
Warm	Evolved judge	0.3709	$\mathbf{0.6683} \pm 0.0459$	**+0.2974**	0.0000
Cold	Self-selection	0.0000	0.6306 ± 0.1352	+0.6306	0.0628
Cold	Fixed-rubric judge	0.0000	0.6199 ± 0.0624	+0.6199	0.0306
Cold	Evolved judge	0.0000	$\mathbf{0.7679} \pm 0.0157$	**+0.7679**	0.0000

5 RESULTS

5.1 WEBSHOP ADAPTATION AND HELD-OUT TRANSFER

Panel A of Table 1 reports higher observed mean rewards under evolved-judge guidance (JET) from both initializations, with a larger mean difference from the cold start.

Warm start. Without test-time evolution, the warm program obtains 0.6256 mean reward. Evolution with the fixed-rubric judge raises this to 0.6589, whereas evolution with the evolved judge reaches 0.6951. The observed mean reward gain is approximately twice that under fixed-rubric guidance, with a final mean difference of 0.0362. Warm-start exact success on the adaptation set is slightly lower with the evolved judge (30.6% versus 31.2%), despite its higher mean reward (Appendix D).

Cold start. The observed mean separation is larger when evolution begins from the unevolved program. Proposer self-selection reaches 0.6137 mean reward, guidance from the fixed-rubric judge reaches 0.6965, and guidance from the evolved judge reaches 0.7797. The evolved-judge condition has a higher observed mean reward by 0.0832 over fixed-rubric guidance and by 0.1660 over self-selection. The optional source constraint has a slightly higher warm-start target mean and a lower cold-start target mean, while maintaining the required source performance (Appendix B).

Under both judges, cold-start evolution reaches higher final mean rewards than warm-start evolution despite beginning from weaker programs. Source optimization biases subsequent rewrites toward source-specific behaviors, making adaptation to the target distribution more difficult. Starting from an unevolved program may therefore give search greater flexibility to develop target-specific behavior.

Held-out target tasks. To test whether the improvements extend beyond the tasks observed during evolution, we freeze each selected program and evaluate it on a disjoint set of previously unseen tasks sampled from the same target distribution. These tasks are not used for evolution, candidate selection, or model development. Panel A of Table 2 shows that the adaptation-set ordering is preserved on the held-out set. From the cold start, evolution with the evolved judge reaches 0.7915 reward, compared with 0.7008 under the fixed-rubric judge and 0.6259 under proposer self-selection. From the warm start, evolution with the evolved judge reaches 0.6951, compared with 0.6700 under the fixed-rubric judge and 0.6387 without test-time evolution. Cold-start evolution with the evolved judge also yields the highest mean held-out exact-success rate (50.6%).

Table 2: WEBSHOP held-out performance and offline controls. (A) Reward mean ± SD and mean success; *no external judge* denotes no evolution from warm and self-selection from cold. (B) Source-holdout MAE and five-campaign mean target MAE and selection regret. Target columns identify the guidance used to generate candidates.

Panel A: Primary held-out program performance

	Warm start		Cold start	
Adaptation guidance	Reward ↑	Success ↑	Reward ↑	Success ↑
No external judge	0.6387 ± 0.0021	27.6%	0.6259 ± 0.0354	24.8%
Fixed-rubric judge	0.6700 ± 0.0374	32.6%	0.7008 ± 0.0499	37.2%
Evolved judge	$\mathbf{0.6951} \pm 0.0052$	**34.2%**	$\mathbf{0.7915} \pm 0.0322$	**50.6%**

Panel B: Offline judge controls

	Prediction error (MAE ↓)			Selection regret ↓	
Scoring judge	Source	Fixed-rubric	Evolved	Fixed-rubric	Evolved
Fixed-rubric judge	0.1576	0.2252	0.2239	0.0081	0.0040
Calibrated fixed-rubric judge	0.1488	0.2227	0.2250	0.0081	0.0040
Evolved judge	0.1168	0.1682	0.1843	0.0000	0.0000
Deterministic evolved judge	0.1153	0.1533	0.1695	0.0000	0.0000

5.2 WHAT THE JUDGE LEARNS

Figure 3(a) shows that source evolution reduces the seed judge's holdout MAE from 0.1480 to 0.1168 (21.1%) and raises Spearman correlation by 0.0351. These gains indicate better reward estimates and more accurate trajectory ranking on source data excluded from development. Figure 3(b) compares both judges on identical trajectories from two auxiliary programs, one adapted under each judge. On partially completed tasks, the evolved judge reduces reward-prediction MAE by 23.4% for the evolved-guided program and 24.7% for the fixed-rubric-guided program. The accuracy advantage therefore also appears on trajectories produced under fixed-rubric guidance. Appendix C shows less underestimation of partial progress, although the fixed-rubric judge remains more accurate on complete failures.

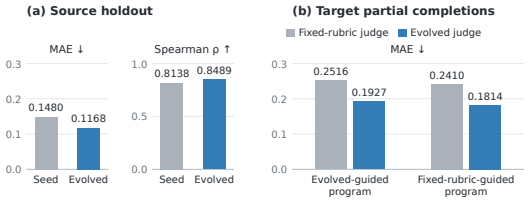


Figure 3: Judge fidelity. (a) Source-holdout MAE and Spearman ρ: seed versus evolved. (b) Partial-reward MAE: fixed-rubric versus evolved on identical trajectories from each auxiliary program. Lower MAE and higher ρ are better.

To examine what judge evolution contributes, we compare the fixed-rubric judge with a source-calibrated version, the evolved judge, and a deterministic version that runs evidence extraction and score adjustment without LLM assessment.

Can score calibration match evolved-judge accuracy? Fitting an isotonic mapping (Zadrozny & Elkan, 2002) on labeled source trajectories reduces source-holdout MAE from 0.1576 to 0.1488, compared with 0.1168 for the evolved judge. The evolved judge also has lower MAE under both target guidance conditions (Table 2, Panel B). Calibration and scoring-control details appear in Appendix E.

Can evolved code predict rewards without fresh LLM assessment? The deterministic evolved judge achieves source-holdout MAE of 0.1153, compared with 0.1168 for the full judge, and lower MAE on both cached target pools (Table 2, Panel B). Comparison with the seed judge's code shows that this independent numerical pathway emerged during source evolution. These results indicate that the evolved code captures useful reward-prediction rules, allowing it to maintain prediction accuracy

on these trajectory sets without a fresh LLM assessment. The full judge additionally generates textual diagnostics for program rewrites, whose contribution is examined separately in Section 5.4.

Request: ivory/blue rug, 2 ft 3 in $\times$ 6 ft, below \$130.
Purchase: matching rug and color, 2 ft $\times$ 8 ft, \$100.
True reward: 0.8333; wrong size prevents exact success.

Fixed-rubric judge
Predicted reward: **0.3200**
Finds the size mismatch but also raises a spurious price concern.

Evolved judge
Predicted reward: **0.8329**
Credits the product, selected color, and price; localizes the size error.

Figure 4: Selected partial-progress example. Both judges score the same reward-scrubbed trajectory; true reward is shown only post hoc. The evolved judge credits satisfied requirements and identifies the remaining error.

How do scoring and diagnosis differ on the same trajectory? Figure 4 compares the judges on a purchase that satisfies the requested product, color, and price but selects the wrong size. The fixed-rubric judge identifies the size mismatch, yet substantially underestimates partial completion and raises an unsupported price concern. The evolved judge credits the satisfied requirements and identifies size as the remaining mismatch. Its more accurate partial-credit estimate is consistent with the lower aggregate partial-reward error in Figure 3(b).

These differences matter because scores and diagnostics serve different roles in program evolution. Scores determine which candidates are retained and also enter proposer feedback; diagnostics provide additional guidance for subsequent revisions (Yuksekgonul et al., 2025; Madaan et al., 2023). Undercrediting completed requirements can undervalue a candidate while an unsupported criticism can direct revisions toward an already satisfied requirement.

5.3 Selection Quality and Candidate Generation

Does improved selection extend to intermediate rounds? At each round within a campaign, each judge selects from the saved candidates generated so far, holding their trajectories fixed. Under fixed-rubric candidate generation, the evolved judge has zero selection regret at every round, whereas the fixed-rubric judge incurs regret at a mean of 17 of 40 rounds per campaign. On candidates generated under evolved-judge guidance, the fixed-rubric and evolved judges incur regret at an average of 18 and 8 rounds per campaign, respectively, and mean campaign maximum regret falls from 0.2538 to 0.0185. The selection advantage therefore extends beyond the final choice under both guidance conditions.

Can better final selection alone explain the performance gain? For each campaign, we hold the saved programs fixed and compare which program each judge selects. The evolved judge achieves zero mean final regret on candidates generated under both fixed-rubric guidance and evolved-judge guidance, while calibration leaves the fixed-rubric judge's mean selected reward unchanged (Table 2, Panel B). Perfect final selection among programs generated under fixed-rubric guidance would raise mean true adaptation reward from 0.6965 to approximately 0.7046, still below 0.7797 under evolved guidance. The mean reward advantage of evolved-judge-guided search would therefore shrink from 0.0832 to 0.0751, closing only about 9.7% of the original difference. Most of this advantage thus reflects differences in the programs generated during search, which better final selection alone cannot recover.

5.4 Judge Baselines and Component Ablations

Figure 5 compares judge development and feedback components through fresh program evolution in a ablation series. Held-out reward is measured on separate target tasks using frozen selected programs.

Does evolving executable judge code offer benefits beyond optimizing its prompt? The source-optimized prompt-only judge holds its executable wrapper fixed. Both source-optimized judges achieve higher observed mean rewards than the unchanged seed, but executable evolution exceeds

prompt optimization by 0.1473 in adaptation reward and 0.1074 in held-out reward. Prompt refinement leaves the judge's evidence extraction and score adjustment code unchanged, whereas executable evolution can revise these components alongside its instructions. The results support optimizing the evaluation procedure as a whole, with the held-out advantage indicating that the resulting gains extend beyond the tasks used for adaptation.

What changes when textual diagnostics are included? With the evolved judge's scoring procedure held fixed, mean adaptation reward is 0.6984 with diagnostics and 0.6579 with scalar feedback only while mean held-out reward is 0.6869 and 0.6608 respectively. These mean differences provide evidence for a benefit from diagnostics. Diagnostics identify requirements and suggest edits beyond the scalar completion estimate.

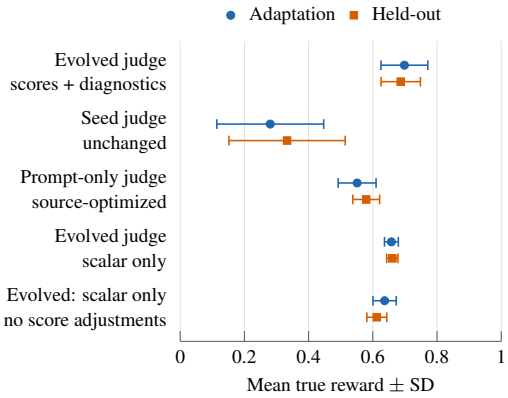


Figure 5: Judge and feedback comparisons (mean ± SD). See Appendix F.

Do numerical score adjustments contribute to program performance? With diagnostics withheld, we bypass we bypass score adjustment while retaining evidence extraction and LLM assessment. Retaining these adjustments yields higher observed mean adaptation and held-out rewards by 0.0211 and 0.0484, respectively. These results suggest that the evolved score adjustment makes scalar feedback more useful for program adaptation beyond the underlying LLM assessment. Because scores determine candidate promotion and also inform subsequent rewrites, the adjustments can influence both which programs are retained and how later candidates are generated.

5.5 PushT: An exact-judge control

We use PushTas an exact-judge control that isolates program search from judge error. The evolved judge recovers the geometric coverage rule in the first source-evolution round and computes block–goal overlap from observable terminal poses (Appendix A.3). Its scalar selection signal therefore equals the the true final coverage metric, although environment-provided reward fields remain withheld.

All adaptation conditions achieve higher observed mean coverage than their starting policies (Table 1, Panel B). From cold, evolved-judge guidance reaches 0.7679 coverage, compared with 0.6199 under fixed-rubric guidance and 0.6306 under proposer self-selection. From warm, it reaches 0.6683 versus 0.6033 under fixed-rubric guidance, an advantage of 0.0650. Because the evolved judge is exact, the gap to fixed-rubric guidance reflects what an approximate judge costs the search. Zero selection regret under evolved-judge guidance means that each campaign selects a policy with the highest true coverage among its generated candidates. Yet mean coverage remains below one even with exact scoring. Further improvement therefore requires generating better policies, rather than making a different final choice among the same candidates.

6 Conclusion and Limitations

JET evolves an executable judge on labeled source trajectories and transfers its frozen scores and diagnostics to test-time program search without target evaluator access. Adapted programs attain higher observed mean rewards, including on unseen target tasks, but limited campaign replication leaves smaller differences, including diagnostic-feedback gains, uncertain. Both benchmarks preserve the evaluation rule; transfer to changed reward criteria or outcomes requiring unobserved information remains untested. Repeated program generation, execution, and evaluation incur inference cost and latency. Broader replication beyond the five primary campaigns and cost analysis remain future work.

AI USE STATEMENT

In this work, we used generative AI tools for feedback on our experimental methodology and on the interpretation of results. We have not used generative AI tools for developing the method or its hypotheses, generating or cleaning data, and mathematical claims, proofs, translation, and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools for readability editing, drafting and restructuring text, and formatting references and LaTeX. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Sections 4 and 5 describe the comparison protocol, metrics, and results. Appendix A collects campaign replication, data splits, search budgets, and execution settings. The remaining appendices provide source-constraint details, judge-error analyses, primary campaign diagnostics, qualitative examples, offline control procedures, and ablation results. Additionally, we will release the code base and artifacts for this research work.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *First Workshop on Foundations of Reasoning in Language Models*, 2025.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Forty-first International Conference on Machine Learning*, 2024.

Evan Frick, Tianle Li, Connor Chen, Wei-Lin Chiang, Anastasios Angelopoulos, Jiantao Jiao, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. How to evaluate reward models for RLHF. In *International Conference on Learning Representations*, volume 2025, pp. 18128–18163, 2025.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International conference on machine learning*, pp. 10835–10866. PMLR, 2023.

Adam Gleave, Michael Dennis, Shane Legg, Stuart Russell, and Jan Leike. Quantifying differences in reward functions. *arXiv preprint arXiv:2006.13900*, 2020.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. DSPY: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Jie Lou, Yaojie Lu, Hongyu Lin, Xinyu Lu, Ben He, Xianpei Han, Debing Zhang, Le Sun, et al. Rethinking reward model evaluation: Are we barking up the wrong tree? In *International Conference on Learning Representations*, volume 2025, pp. 58769–58790, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Bingchen Miao, Yang Wu, Minghe Gao, Qifan Yu, Wendong Bu, Wenqiao Zhang, Yunfei Li, Siliang Tang, Tat-Seng Chua, and Juncheng Li. Boosting virtual agent learning and reasoning: A step-wise, multi-dimensional, and generalist reward model with benchmark. *arXiv preprint arXiv:2503.18665*, 2025.

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

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Siddharth Reddy, Anca Dragan, Sergey Levine, Shane Legg, and Jan Leike. Learning human objectives by evaluating hypothetical behavior. In *International Conference on Machine Learning*, pp. 8020–8029. PMLR, 2020.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. Rethinking the evaluation of harness evolution for agents. In *COLM 2026 The 2nd Workshop on Lifelong Agents: Learning, Aligning, and Evolving*, 2026.

Jiayi Weng. Learning beyond gradients. `https://trinkle23897.github.io/learning-beyond-gradients/`, May 2026. Blog post.

Yu Xia, Jingru Fan, Weize Chen, Siyu Yan, Xin Cong, Zhong Zhang, Yaxi Lu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. AgentRM: Enhancing agent generalization with reward modeling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19277–19290, 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022b.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025.

Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 694–699, 2002.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (stop): Recursively self-improving code generation. In *First Conference on Language Modeling*, 2024.

Guibin Zhang, Leo Lu, Fangzhou Xie, Kang Zhu, Junhao Wang, Zhifei Xie, Zhaochen Yu, Zihang Liu, Zhongxiang Sun, Qiankun Li, et al. Jit-agent: Scaling harness intelligence via just-in-time harness evolution. *arXiv preprint arXiv:2608.25593*, 2026.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025a.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Seyed Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *International Conference on Learning Representations*, volume 2025, pp. 12476–12505, 2025b.

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. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 80569–80611. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/zhuge25a.html.

A Experimental Settings

A.1 Independent Campaigns and Search Settings

Each primary test-time adaptation condition is evaluated using five independent campaign seeds. A campaign is a complete program-evolution run from a specified initialization; evolution rounds and candidate proposals are steps within that run. Within each campaign, the judge remains frozen and environment-provided evaluator outputs are used only for post-hoc measurement. Dataset-shuffling seeds are distinct from campaign replication.

Table 3 collects the primary test-time search settings. For each selected program, reward and success are first averaged over its evaluation tasks. Initial reward, improvement, oracle-best reward, and selection regret are computed within each campaign before averaging across campaigns. Tables 1 and 2 (Panel A) report campaign means, with SDs for selected-program rewards and coverage where available. Panel B reports source-holdout prediction error and campaign-averaged target prediction error and selection regret. The code proposer used is GPT-5.5 through Codex, with medium reasoning effort while both solver's and judge's LLM calls use GPT-OSS-120B, served locally. Figure 5 and Table 7 report campaign means and standard deviations for the judge and feedback comparisons. Offline scoring and selection procedures are specified in Appendix E.

Table 3: Primary test-time search settings. These settings are shared by adaptive conditions within each benchmark and initialization. Baselines without test-time evolution perform no search.

Setting	WebShop	PushT
Independent campaign seeds per condition	5	5
Evolution iterations per campaign	40	40
Maximum environment steps per trajectory	100	100
Initializations	Warm and cold	Warm and cold

A.2 WebShop Data and Execution Details

This subsection describes the primary WebShop experiments.

Experiments use the full catalog of 1,181,436 products and an 11,674,685-goal synthetic attribute-generated universe, with upstream task-shuffle seed 233. The adaptation set contains 500 non-clothing tasks: 300 garden, 81 beauty, 71 electronics, and 48 grocery. The held-out target set has the same domain composition and disjoint session IDs; it is never exposed to the proposer, judge, or selection rule.

Source data comprise 500 clothing training tasks, 300 validation tasks, and 300 holdout tasks. The evolved judge uses a fixed trajectory dataset generated by the seed program on these tasks; holdout trajectories are used only for reporting. Judge development uses 20 proposer rounds, training-MAE candidate selection, and private validation-MAE final selection. Warm programs are evolved on the

500 source training tasks, and the optional source constraint queries a separate fixed set of 50 source tasks.

Observation scrubbing. Trajectories supplied to the proposer and judge retain the task instruction, ordinary product observations, actions, and available interaction targets. Environment-provided final and per-step rewards, coverage, success indicators, reward breakdowns, and evaluator-only goal annotations are excluded. Scrubbing removes evaluator-derived information from both structured trajectory fields and rendered observation text, including the post-purchase final observation. The retained task instruction and ordinary product information remain available for judging completion; target evaluator outputs and evaluator-only annotations are not exposed.

The judge emits a continuous `near_success` score, a binary `passed` field, and textual failure, reason, and repair fields. Mean `near_success` is used as J_{score} for candidate promotion; the binary field is reported but does not determine promotion, while diagnostics remain visible to the proposer. The selected judge combines an LLM call with deterministic calibration logic.

The primary proposal protocol permits use of information and vocabulary visible in the target interaction, but prohibits explicit task identifiers, hidden evaluator access, and external campaign state. The resulting program may specialize to the observed deployment environment. No target evaluator output is returned to the proposer, judge, or promotion rule in any reported test-time campaign.

A.3 PushT Task and Protocol Details

The agent program is an executable Python control policy that maps observed state to a two-dimensional target position. The proposer and judge observe block and goal poses; stored coverage, success, and reward fields are withheld. Final coverage is the intersection area of the final block and goal shapes divided by the goal area. The fixed-rubric judge estimates completion with an LLM rubric, while the evolved judge reconstructs overlap from visible geometry. Let $B_T \subset \mathbb{R}^2$ denote the region occupied by the block at its terminal pose, and let $G \subset \mathbb{R}^2$ denote the goal region. Final coverage is

$$C_T = \frac{\text{Area}(B_T \cap G)}{\text{Area}(G)} \in [0, 1]. \tag{3}$$

Source tasks start near the goal without damping; target tasks use larger initial displacement and damping 0.2. The warm policy is developed on labeled source tasks. The frozen warm policy obtains 0.3709 coverage on the target set, while the cold policy obtains zero. Target adaptation uses the settings in Table 3 and retains the highest-scoring eligible policy.

The fixed-rubric judge has training and validation coverage MAE of 0.0624 and 0.0742. The evolved judge computes the final polygon intersection fraction and obtains 0.0000 MAE on both splits. We treat this condition as an exact-judge control: policy behavior must adapt to changed dynamics, while the reconstructed coverage rule remains unchanged.

B Optional Source-Capability Preservation

B.1 Constraint Definition

The core method requires no source-evaluator queries during target adaptation. When retaining a verified level of source capability is desirable, the framework can optionally impose a labeled source constraint without using target evaluator outputs. Let $G \subset \mathcal{T}_{\text{src}}$ be a source check set, $Q_G(\pi)$ its source-evaluator score, b a fixed reference floor, and $\epsilon \geq 0$ a tolerance. In the reported experiments, b is the initial program's score on the source check set. A candidate π is eligible only if

$$Q_G(\pi) \geq b - \epsilon. \tag{4}$$

The constraint anchors otherwise reward-hidden target adaptation to an evaluator-measured source capability. It guarantees only aggregate performance on the fixed source check set, not target performance or per-task source non-regression. Its task set, metric, reference policy, and tolerance are environment-specific experimental choices rather than required components of evaluator transfer. For source-constrained campaigns, reported selection regret can include the cost of excluding source-ineligible candidates; it should not be interpreted solely as judging error.

C Detailed Error Decomposition

Judge-quality analyses report MAE and Spearman correlation. For completeness, the seed judge and evolved judge obtain training MAE of 0.1685 and 0.1038, respectively, and validation MAE of 0.1637 and 0.0982. The main paper reports prediction error on the source holdout excluded from judge development.

Table 4 summarizes source-holdout fidelity and partial-reward accuracy.

Table 4: Judge fidelity. Source rows compare seed and evolved judges; target rows compare fixed-rubric and evolved judges on auxiliary partial-reward trajectories.

Metric / trajectory source	Reference	Evolved
Source holdout: seed reference		
MAE ↓	0.1480	**0.1168**
Spearman ρ ↑	0.8138	**0.8489**
Target partial-reward MAE ↓		
Fixed-rubric reference		
Evolved-guided program	0.2516	**0.1927**
Fixed-rubric-guided program	0.2410	**0.1814**

The following matched cross-scoring analysis uses saved trajectories from auxiliary WebShop campaigns rather than the primary program matrix. Within each row, both frozen judges score the identical 500 reward-scrubbed trajectories, so the comparison isolates prediction on a fixed trajectory set.

Table 5: Auxiliary matched target cross-scoring. Panel A reports aggregate prediction and ranking accuracy; Panel B decomposes error by ground-truth reward regime. Bias is mean prediction minus true reward. Lower MAE and higher Spearman ρ are better.

Panel A: Aggregate accuracy and ranking

Trajectory set	True mean	Fixed-rubric MAE	Evolved MAE	Fixed-rubric ρ	Evolved ρ
Program guided by evolved judge	0.7373	0.2824	**0.1811**	0.5069	**0.6613**
Program guided by fixed-rubric judge	0.7025	0.2377	**0.1725**	0.6055	**0.7029**

Panel B: Error decomposition by reward regime

Trajectory set	Regime	N	Fixed-rubric MAE	Fixed-rubric bias	Evolved MAE	Evolved bias
Program guided by evolved judge	Zero	15	**0.1500**	+0.1500	0.3954	+0.3954
	Partial	287	0.2516	-0.1741	**0.1927**	-0.0042
	Exact	198	0.3370	-0.3370	**0.1480**	-0.1480
Program guided by fixed-rubric judge	Zero	22	**0.1523**	+0.1523	0.3006	+0.3006
	Partial	297	0.2410	-0.1521	**0.1814**	+0.0139
	Exact	181	0.2425	-0.2425	**0.1423**	-0.1423

Most target outcomes are partial or exact. The evolved judge's aggregate advantage comes from these strata, while the fixed-rubric judge better recognizes the small number of true-zero cases. This trade-off is compatible with useful dense optimization and occasional severe false optimism.

D PRIMARY CAMPAIGN DIAGNOSTICS

Search trajectories. Figure 1 compares the primary cold-start searches guided by the fixed-rubric and evolved judges. Each campaign uses 40 valid evolution rounds, and 500 adaptation tasks, without a source constraint. Within each campaign, candidate rewards are first averaged over the adaptation tasks. Panel (a) then averages the retained programs' rewards across campaigns at each round. Panel (b) first identifies the highest mean reward attained within each campaign's generated candidates through that round, including rejected candidates, and then averages these values across campaigns. Hidden rewards are used only for post-hoc analysis and never for online selection. Round 0 is evaluated separately in each campaign. Each curve averages independent target-adaptation campaigns; Held-out target performance is reported separately in Table 2, Panel A.

Table 6 reports additional diagnostics from the primary adaptation campaigns. Oracle-best and regret are computed within each campaign's generated candidates before averaging across campaigns.

Table 6: Search diagnostics averaged across the WEBSHOP campaigns. Selection score is the judge's predicted score under judge guidance and the proposer's self-score under self-selection. Oracle-best, regret, and exact success use hidden reward only post hoc. Success is measured on the adaptation tasks.

Condition	Selection score	Oracle-best	Regret	Success
Warm + fixed-rubric judge	0.5059	0.6788	0.0198	31.2%
Warm + evolved judge	0.6487	0.6951	0.0000	30.6%
Warm + evolved judge + source constraint	0.6550	0.7145	0.0111	35.2%
Cold + self-selection	0.7800	0.6258	0.0121	23.0%
Cold + fixed-rubric judge	0.5701	0.7046	0.0081	35.6%
Cold + evolved judge	0.7489	0.7797	0.0000	49.2%
Cold + evolved judge + source constraint	0.6966	0.7100	0.0000	35.8%

E OFFLINE JUDGE CONTROL PROCEDURES

Campaigns and accounting. The target controls in Table 2, Panel B, rescore saved trajectories from five primary cold-start WEBSHOP campaigns per guidance condition, without a source constraint. Each campaign evaluates the initial program and 40 generated candidates on the same 500 adaptation tasks: 41 candidate evaluations and 20,500 trajectory records per campaign. All scoring judges receive identical reward-scrubbed trajectories within each campaign; agent execution is held fixed. Original predictions are retained when available, with fresh predictions obtained for cross-scoring. Target MAE is computed within each campaign and then averaged over its guidance condition's campaigns; source MAE is computed on the fixed source holdout.

Final selection and reported values. Within each campaign, each judge selects the candidate with the highest mean predicted score across the 500 tasks; ties favor the earlier candidate. Audited selected reward, oracle-best reward, and regret are computed within the campaign before averaging. Table 2, Panel B, is the main report of cross-scoring prediction and selection results. Primary selected rewards and within-campaign regret are reported in Table 1; the final-selection accounting in Section 5.3 uses those displayed means.

Source-fitted isotonic calibration. Let s_i be the fixed-rubric judge's score and r_i the source evaluator reward for training trajectory i. We fit a non-decreasing mapping by unweighted least-squares isotonic regression (Zadrozny & Elkan, 2002):

$$\hat{g} \in \underset{g \text{ nondecreasing}}{\arg\min} \sum_{i=1}^{500} \big(g(s_i) - r_i\big)^2.$$

Predictions outside the observed training-score range use the fitted boundary values. The calibrated fixed-rubric judge returns $\hat{g}(s)$ without changing trajectory interpretation or diagnostics. No validation, holdout, or target labels are used to fit the map or select its settings. Although the mapping preserves trajectory-level score order up to ties, applying a nonlinear transformation before averaging

can change candidate rankings. The fitting objective is squared error, whereas source judge evolution optimizes MAE.

Deterministic scoring. The deterministic evolved judge extracts trajectory features, computes a structural score, and applies the evolved postprocessing rules with LLM-generated fields left empty. It retains the extraction rules, scoring tables, calibration functions, and missing-field defaults, and obtains no fresh LLM judgment. LLMs are still used during judge development and to generate the agent trajectories. This intervention uses the frozen evolved judge selected by source-validation MAE for the primary WEBSHOP experiments. Comparing its code with the seed judge confirms that the independent numerical scoring pathway appeared during source evolution. LLM-dependent inputs are removed together.

Offline prefix selection. For campaign c, let $\mathcal{P}_{c,t}$ contain its candidates generated through iteration t. Define

$$B_{c,t} = \max_{\pi \in \mathcal{P}_{c,t}} \bar{R}_c(\pi), \qquad S_{c,t}(J) = \bar{R}_c\left(\arg\max_{\pi \in \mathcal{P}_{c,t}} \bar{J}_c(\pi)\right),$$

where bars denote means over the fixed adaptation tasks and ties follow the earlier-candidate rule. Prefix regret is $B_{c,t} - S_{c,t}(J)$. Within each campaign, we count positive-regret prefixes among its 40 post-initialization stages and compute its maximum prefix regret; each statistic is averaged across five campaigns.

The main paper reports the mean number of prefixes with positive regret (out of 40 per campaign) and the mean of the campaign-specific maximum regrets. These comparisons describe prediction and selection on saved candidates; they do not isolate the effects of promotion, diagnostics, and proposal randomness on candidate generation.

F JUDGE BASELINES AND COMPONENT ABLATION RESULTS

All conditions use five independent target-adaptation campaigns starting from the same unevolved agent program (cold start). We retain the primary WebShop task sets: 500 clothing source-training tasks, 300 source-validation tasks, 300 source-holdout tasks, 500 non-clothing adaptation tasks, and 500 disjoint held-out target tasks. Prompt-only optimization and executable evolution start from the same judge implementation and use identical source trajectories, validation procedures, and budgets of 20 proposer rounds. Training MAE determines the evolutionary parent, and private source-validation MAE selects the final judge; source-holdout data are excluded from development. The selected judges remain frozen during target adaptation. Each campaign uses 40 search rounds, and at most 100 environment steps per trajectory. This separate ablation series retains the primary data and search protocol while varying the judge-development procedure and the availability of diagnostics or numerical score adjustments.

Table 7 gives the numerical results plotted in Figure 5. Values are mean true rewards and standard deviations across campaigns for each condition. Held-out evaluation uses frozen selected programs on separate target tasks.

Table 7: Judge baselines and component ablations. Mean true reward ± standard deviation across campaigns for each condition.

Judge and feedback condition	Adaptation	Held-out
Evolved judge: scores + diagnostics	$\mathbf{0.6984} \pm 0.0730$	$\mathbf{0.6869} \pm 0.0613$
Unchanged seed judge	0.2805 ± 0.1664	0.3329 ± 0.1811
Source-optimized prompt-only judge	0.5511 ± 0.0591	0.5795 ± 0.0417
Evolved judge: scalar feedback only	0.6579 ± 0.0216	0.6608 ± 0.0175
Evolved judge without score adjustments: scalar feedback only	0.6368 ± 0.0362	0.6124 ± 0.0311