ENVOPT: CLOSED-LOOP ENVIRONMENT OPTIMIZATION FOR AGENT POST-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Data-centric recursive self-improvement (RSI) seeks to improve a model by repeatedly turning its evolving behavior into new training data. For tool-using agents, sustaining this process is more difficult because the data are executable environments: each round must preserve a coherent and tight connection from observed behavior to training intent, executable realization, and realized learning outcomes. Our analysis finds that even a capable agent frequently loses this connection, showing an incomplete understanding of model behavior, unfaithful implementation of intended environments, and ineffective use of feedback across iterations. To address these challenges, we introduce ENVOPT, a framework that organizes environment evolution around traceable training targets. It combines a traceable behavioral map to expose recurring patterns, iterative environment construction to faithfully realize training targets as executable environments, and target-conditioned context refinement to facilitate accumulated feedback utilization. This turns environment generation from a sequence of isolated attempts into a closed-loop data improvement process that adapts as the target model changes. Extensive experiments on τ^2-Bench, BFCL-v4, ACEBench-Agent, and AppWorld show that ENVOPT yields substantial improvements in the target model's performance, while the vanilla agent baseline shows no consistent improvement trend.

1 INTRODUCTION

Recursive self-improvement (RSI) asks whether AI systems can use evidence from one development cycle to improve the next. MetaRSI (Tan et al., 2026) identifies data, harness, and model as three writable surfaces for improvement. Among these, autonomously developing training data and using it to update model parameters offers a key pathway toward stronger intelligence. We therefore focus on the interaction between the data and model surfaces, where a data agent constructs and adapts training data as the target model evolves. RSIBench-Data (Meng et al., 2026) shows that agents can discover useful data strategies but struggle to improve them consistently across training runs, exposing a discovery–reliability gap. This raises a sharper question: when the target model changes, can feedback from one training round reliably guide what data to construct next?

We study this question in the context of training agents to use tools more effectively, as tool use is a fundamental capability for modern agents. Training such agents extends beyond generating static samples because training data take the form of executable instances that define state, tool behavior, tasks, and outcome verification. Systems for executable environment synthesis and scaling, including AgentScaler, EnvScaler, ScaleEnv, Agent World Model, and EnvFactory (Fang et al., 2025; Song et al., 2026; Tu et al., 2026; Wang et al., 2026; Xu et al., 2026), show that such resources can be constructed automatically. These works establish the feasibility of automated environment construction, while data-centric RSI further requires the agent to decide how these environments should evolve as the target model changes. We instantiate this process in a stateful, multi-round online-RL setting, where a target model is iteratively updated using new data produced by a data agent. Starting from an empty environment pool, the agent repeatedly analyzes the current target model's behavior, constructs executable training instances, submits them to a fixed external service for online-RL training and evaluation, and uses the resulting feedback to guide the next round. With each round continuing from the updated model, the training data and target policy evolve together over time. Reliable recursive improvement is therefore not a one-shot data-generation problem: it requires sustaining a coherent data loop as both continue to evolve. Our experiments show that even a modern agent with

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.

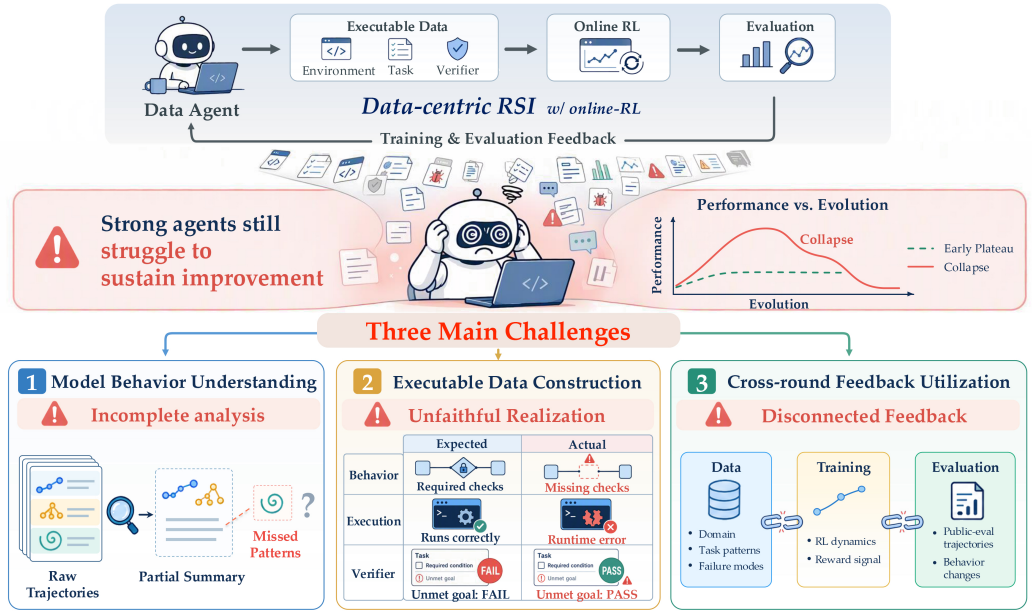


Figure 1: Three challenges in closing the data-centric RSI loop: incomplete behavioral analysis, unfaithful realization of executable training instances, and disconnected data–training–evaluation feedback.

strong reasoning and coding capabilities (Codex with GPT-6 Astra (OpenAI, 2026b)) struggles to sustain improvement across rounds.

Based on our experiments and observations, we identify three key stages in closing the data improvement loop and a recurring failure mode at each. Model behavior understanding is limited by **incomplete analysis** of available traces, yielding a partial or biased view of the target model. Executable data construction exhibits **unfaithful realization** when data instances do not exercise the intended training target. Cross-round feedback utilization is disrupted by **disconnected feedback**, which prevents the agent from relating constructed data to training dynamics and subsequent target behavior.

To address these challenges, we introduce ENVOPT, a framework that organizes environment evolution around traceable training targets through three corresponding components. To identify patterns in target model behavior, a *traceable behavioral map* consolidates evidence across trajectories while preserving links to the source cases. To faithfully realize each training target as executable data, *iterative environment construction* uses a planning, implementation, and refinement loop to bridge the last mile from target specification to logically consistent and behaviorally aligned environments. To close the feedback loop across rounds, *target-conditioned context refinement* organizes relevant data, observed training dynamics, and subsequent evaluation outcomes around each training target. Experiments on four tool-use benchmarks demonstrate that ENVOPT sustains multi-round improvement. Averaged across the four benchmarks, it improves performance by $+14.94$ and $+7.49$ percentage points on Qwen3-4B and Qwen3-8B. Its performance shows a clear overall upward trend across iterations, while vanilla agents fluctuate without sustained improvement. Further analyses show improved task and failure coverage, more reliable and target-aligned environments, and higher-quality realized RL signals.

2 RELATED WORK

Data-centric RSI. RSIBench-Data studies agents that revise training-data strategies for a separate target model under fixed training, evaluation, and resource controls (Meng et al., 2026). Autodata develops an agentic data scientist that iterates between data creation and quality assessment, including meta-optimization of the data scientist itself (Kulikov et al., 2026). DataMaster combines data

discovery and transformation with tree search and cumulative memory while holding the learning algorithm fixed (Du et al., 2026). ENVOPT studies this research role for executable training environments: a researcher starts with an empty environment library and uses behavioral evidence to guide successive rounds of environment revision and online RL. Its focus is the evidence supporting those research decisions.

Executable environment synthesis for tool-calling agents. AgentScaler, EnvScaler, and ScaleEnv automate the construction of executable tools, state, and verifiable tasks (Fang et al., 2025; Song et al., 2026; Tu et al., 2026). Agent World Model scales database-backed synthetic environments for RL, while EnvFactory combines executable environment construction with topology-aware task synthesis (Wang et al., 2026; Xu et al., 2026). FACET grounds task instructions, solutions, and verifiers in shared executable state to preserve source intent and cross-artifact consistency (Shi et al., 2026). The ACE framework and WRIT further examine data accuracy, learner-relative complexity, diversity, and the evidence required for individual tool-use decisions (Zeng et al., 2026; Gu et al., 2026). These works establish construction and validation principles that ENVOPT builds on; our emphasis is using source-traceable behavioral analysis and actual-training feedback to decide which mechanisms to construct or revise.

Iterative RL and environment adaptation. PAIRED and Prioritized Level Replay adapt environment generation or selection using regret and learning-potential signals (Dennis et al., 2020; Jiang et al., 2021). For tool-use agents, RODS uses online reward statistics to synthesize structurally aligned tasks and update a replay pool (Fang et al., 2026). EnvHarness adapts existing environments from policy trajectories while retaining their original verifiers (Huang et al., 2026). SPADE trains a shared environment-designer and solver model through self-play, using hint-based regret, corpus grounding, and environment memory (Liu et al., 2026). Agent-World and AgentOmnia also use behavioral diagnosis to guide targeted task synthesis and self-evolution (Dong et al., 2026; Jiang et al., 2026). ENVOPT uses an external researcher to revise independent environments without the target backend or verifier. It connects the source evidence behind a diagnosis to realized program versions, training tasks, RL observations, and fixed-target evaluation, supporting inspection of what each intervention actually delivered.

3 AGENTS AS DATA RESEARCHERS: IDENTIFYING CHALLENGES IN DATA-CENTRIC RSI

We instantiate data-centric RSI as an iterative loop between a data agent and a target model. The agent constructs executable environments and tasks for training and uses feedback to guide subsequent data decisions. We first formalize this loop and then examine its key challenges.

Formalizing the data loop. We model an executable environment as a partially observable interaction process

$$e = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}).$$

Here, $\mathcal{S}$ denotes the state space over structured environment data (e.g., database records), and $\mathcal{A}$ contains the executable actions exposed through tool calls. The transition function $\mathcal{T}(s' \mid s, a)$ determines how an action updates the state, while the observation function $\mathcal{O}(o \mid s', a)$ returns the tool output or user response visible to the target model.

At round t, the data agent constructs a set $D_t = \{(e_j, x_j, v_j)\}_{j=1}^{n_t}$ of n_t executable training instances. The task x_j defines the goal and initial conditions, while the verifier v_j evaluates task completion. The data generation process is conditioned on a research context C_t, which typically includes the agent's execution history, accumulated memory, prior feedback, external examples, and the current pool of environments and tasks. Formally,

$$D_t = \mathrm{DataAgent}(C_t). \tag{1}$$

Let π_t denote the target model at the start of round t. D_t is submitted to a fixed external service, which updates π_t through online RL and evaluates the resulting model:

$$(\pi_{t+1}, F_t) = \mathrm{TrainEval}(\pi_t, D_t), \tag{2}$$

where π_{t+1} is the updated target model and F_t denotes the resulting feedback, comprising training dynamics and evaluation results. Finally, the generated data and resulting feedback are incorporated

into the research context for the next round:

$$C_{t+1} = \text{UpdateContext}(C_t, D_t, F_t). \tag{3}$$

The loop starts from an empty environment pool, and each subsequent round continues from the updated model π_{t+1}, allowing the generated data and target model to evolve together. To better understand this process, we analyze three critical aspects of the loop and identify common failure modes in each, focusing on how the data agent uses C_t to understand model behavior, realizes the selected training targets as executable instances in D_t, and incorporates F_t into C_{t+1}.

3.1 Model Behavior Understanding: Incomplete Analysis

Using C_t to select suitable training targets requires a comprehensive understanding of the target domain, task patterns, model behavior, and recurring failures. To evaluate this capability, we measure the behavioral coverage of Codex with GPT-6 Astra when analyzing raw trajectories. We give the agent full access to the trajectories and ask it to produce a comprehensive report on the observed task patterns and failure modes. To promote a fair comparison and reduce evaluator self-preference, we use Gemini 3.8 Flash (Google DeepMind, 2026), from a different model family, to construct evidence-grounded reference units and evaluate the report against them. Appendix C.1 provides the full evaluation details. Across $1{,}935$ trajectories from four benchmarks, the report achieves average task-pattern and failure-mechanism coverage of 78.1% and 58.8%, respectively. These results show that the agent may overlook important cases and therefore miss the behavioral evidence needed to determine what training data the target model requires (Figures 6 and 9).

3.2 Executable Data Construction: Unfaithful Realization

Executable data construction requires environment behavior, task requirements, and verifier feedback to jointly and faithfully realize the intended training target. Yet executability alone does not guarantee faithful realization. The cases in Appendices B.3 and C.3 illustrate three failure modes concerning what behavior is trained, how the interaction operates, and how task completion is judged. *Missing task mechanisms* allow the model to bypass the intended behavior. *Faulty interaction logic* causes responses, state transitions, or termination conditions to violate the task rules. *Misaligned verifiers* produce rewards that do not accurately reflect the model's progress or satisfaction of the task requirements. These failures produce training experience that deviates from the intended target, making subsequent feedback unreliable for guiding environment evolution.

3.3 Cross-round Feedback Utilization: Disconnected Feedback Across Data, Training, and Evaluation

Producing suitable, learnable, and capability-aligned tasks requires tight connections among targeted data creation, actual training, and subsequent evaluation outcomes. By integrating information from these three stages, the agent can identify which capabilities require improvement and adjust its data generation strategy accordingly. However, even strong agents struggle to identify the most suitable directions and strategies for data generation (Figure 7 shows a real case where task difficulty remains unchanged despite saturated rewards). We observe that only 21.0% of the training tasks generated by the vanilla Codex agent exhibited reward variation across rollouts (Figure 4). This limited proportion suggests that feedback was not effectively translated into learnable training tasks, leaving much of the available training signal underutilized.

4 Method

EnvOpt instantiates the data-research loop in Equations 1–3 (Figure 2) through 3 components corresponding to the challenges in Section 3. A *traceable behavioral map* (TBM) preserves task requirements and failure mechanisms with links to source trajectories. Guided by the research context C_t, *iterative environment construction* (IEC) produces the environment–task–verifier set D_t through planning, implementation, and independent review. *Target-conditioned context refinement* (TCR) organizes training targets, actual RL experience, and evaluation feedback to refine this context for the next iteration. The fixed training-and-evaluation service updates the target model and returns feedback for the next iteration.

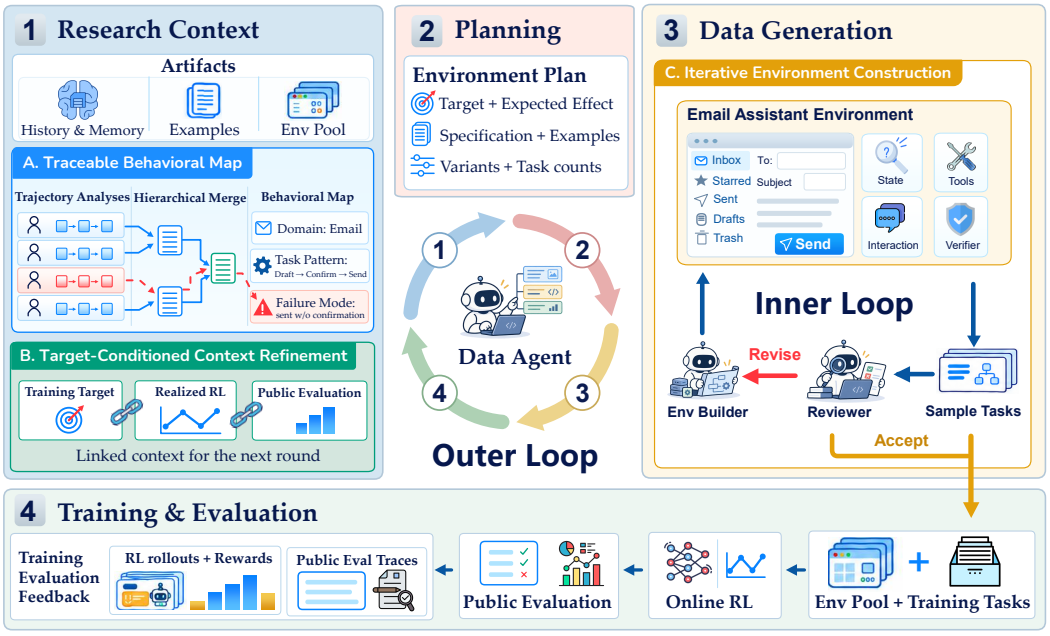


Figure 2: The evolution loop of ENVOPT. The data agent analyzes model behavior, plans and constructs training environments, and uses feedback from online RL and public evaluation to guide the next iteration.

4.1 TRACEABLE BEHAVIORAL MAP

The map separates a persistent evidence layer from its compact reading view. An LLM performs extraction, source review, merging, and refinement. Let $T_{t,i}$ contain the raw trajectories for task i, including instructions, tool definitions, and observed interactions. Extraction and source review produce a set of source-grounded mechanism records:

$$\mathcal{M}_t = \bigcup_i \text{ExtractReview}(T_{t,i}). \tag{4}$$

Each record links an environment mechanism, task requirement, or observed failure or recovery to its source and applicable conditions. Bounded hierarchical merging builds a draft by factoring shared premises while retaining distinct actors, scopes, dependencies, and recovery branches:

$$R_t^{\text{draft}} = \text{FactorMerge}(\mathcal{M}_t). \tag{5}$$

For refinement, we randomly sample 10% of the mechanism records to form a review subset $\mathcal{M}_t^{\text{check}} \subseteq \mathcal{M}_t$. Checking only this compact subset makes refinement cost-efficient by avoiding reanalysis of complete trajectories. An LLM checks whether the draft preserves the sampled mechanisms and their applicable conditions. Repair corrects conflicting report entries, and Supplement consolidates missing distinctions identified in the sampled records:

$$\begin{aligned} \bar{R}_t &= \text{Repair}(R_t^{\text{draft}}, \mathcal{M}_t^{\text{check}}), \\ R_t &= \bar{R}_t \oplus \text{Supplement}(\mathcal{M}_t^{\text{check}}, \bar{R}_t). \end{aligned} \tag{6}$$

Here $\oplus$ appends the missing distinctions without rewriting preserved content. An already-covered record must instead be matched to an existing passage.

The report retains the four fields *domain, environment, task behavior, failure behavior*. It enters the research context C_t with links back to original trajectories. Section 5.3 evaluates the behavioral coverage of the resulting reports. Appendix D.2 provides an example of the generated report.

4.2 ITERATIVE ENVIRONMENT CONSTRUCTION

Using the research context C_t, which integrates behavioral analysis, training feedback, and accumulated experience, the data agent selects environments to create or revise and allocates training tasks.

It specifies each environment's objective, requirements, and expected behaviors, then delegates implementation to a persistent Env Builder subagent. Each builder produces candidate (e, x, v) triples aligned with the specification. The current target model then performs probe rollouts on representative tasks. An independent review agent examines the implementation, generated tasks, execution checks, and probe rollouts to assess implementation correctness, target alignment, and potential task difficulty. It produces a review report with a decision to accept the candidate or request refinement, together with actionable feedback. Implementation issues return to the same builder for revision and re-review, while specification or difficulty issues return to the data agent for subsequent data decisions. Finally, this process produces a batch of quality-checked training data D_t for RL. By aligning environment behavior and reward feedback with the intended capability targets, iterative refinement provides more reliable learning signals for the target model.

4.3 Target-Conditioned Context Refinement

Effective data evolution requires interpreting training and evaluation feedback in relation to the capabilities the generated data was intended to improve. We therefore organize the research context around explicit capability targets. Before proposing environment changes, the data agent records the target requirement and expected effect based on observed model behavior. These provide a shared reference for interpreting subsequent feedback and deciding what data to generate next.

We then perform version-aware evidence alignment. Environment versions and training-task IDs link each target to its accepted instances in D_t and actual RL groups and rollouts. Associated evaluation tasks connect these training records to performance on the targeted capability. The resulting target-conditioned evidence view distinguishes planned, accepted, and actually trained data while bringing together three complementary perspectives:

$$\underbrace{\text{Training target}}_{\text{expected capability}} \longleftrightarrow \underbrace{\text{Actual training}}_{\text{RL dynamics}} \longleftrightarrow \underbrace{\text{Target evaluation}}_{\text{related performance}}. \tag{7}$$

Context refinement proceeds through expectation–observation comparison. The data agent jointly examines training rewards, within-task reward contrast, and related evaluation outcomes and trajectories against the expected effect. It assesses whether the generated data provided useful learning signals and whether the targeted behavior improved, recording supporting observations, mismatches, and unresolved questions in C_{t+1}. These accumulated findings guide revisions to environment mechanisms, task difficulty, and sampling, as well as decisions to retain or pause environments. Each iteration thus refines the research context used to plan the next training intervention.

5 Experiments

5.1 Experimental Setup

Evaluation and training. We evaluate all methods on τ^2-Bench (Barres et al., 2026), BFCL-v4 (multiturn) (Patil et al., 2025), ACEBench-Agent (Chen et al., 2025), and AppWorld (Trivedi et al., 2024). The first three benchmarks use 1:1 public and private splits, while AppWorld uses its train split for public evaluation and its test-normal split for private evaluation. Results on the public splits guide environment evolution, whereas the private splits remain hidden throughout the evolution process and are used only for final performance comparisons. We train the target model using GRPO (Shao et al., 2024) and report the performance of the final checkpoint produced by the complete evolution process on the held-out private evaluation splits. Reported benchmark scores are averaged over 3 evaluation repeats.

Implementation and baselines. Our target models are Qwen3-4B-Instruct-2507 and Qwen3-8B (Yang et al., 2025), with reasoning enabled only for the latter. We instantiate the data agent, DataAgent, with Codex and provide it with access to the GPT-5.5 (OpenAI, 2026a) API and four A100 GPUs during evolution. We compare against two baselines: vanilla Codex acting as an autonomous data agent (`Codex-Vanilla`) and online RL using a fixed environment set generated once by Codex without subsequent revision (`Fixed-Env`). All data-agent configurations use Codex powered by GPT-6 Astra with high reasoning effort. Further experimental details are provided in Appendix D.

Table 1: Test performance (%). Deltas from None (pp) are green for gains and red for drops. Avg. averages all four benchmarks.

Method	τ^2-Bench	BFCL-v4	ACEBench-Agent	AppWorld	Avg.
Qwen3-4B-Instruct-2507					
None	31.89	25.42	41.11	8.93	26.84
`Codex-Vanilla`	21.58 –10.31	25.75 +0.33	18.89 –22.22	13.69 +4.76	19.98 –6.86
`Fixed-Env`	27.58 –4.32	33.17 +7.75	45.00 +3.89	5.56 –3.37	27.83 +0.99
ENVOPT	**50.60** +18.71	**42.08** +16.67	**52.22** +11.11	**22.22** +13.29	**41.78** +14.94
Qwen3-8B (reasoning enabled)					
None	28.54	39.50	43.89	17.86	32.45
`Codex-Vanilla`	14.39 –14.15	43.67 +4.17	50.00 +6.11	6.75 –11.11	28.70 –3.75
`Fixed-Env`	31.41 +2.87	42.58 +3.08	46.11 +2.22	22.62 +4.76	35.68 +3.23
ENVOPT	**37.41** +8.87	**46.67** +7.17	**51.67** +7.78	**24.01** +6.15	**39.94** +7.49

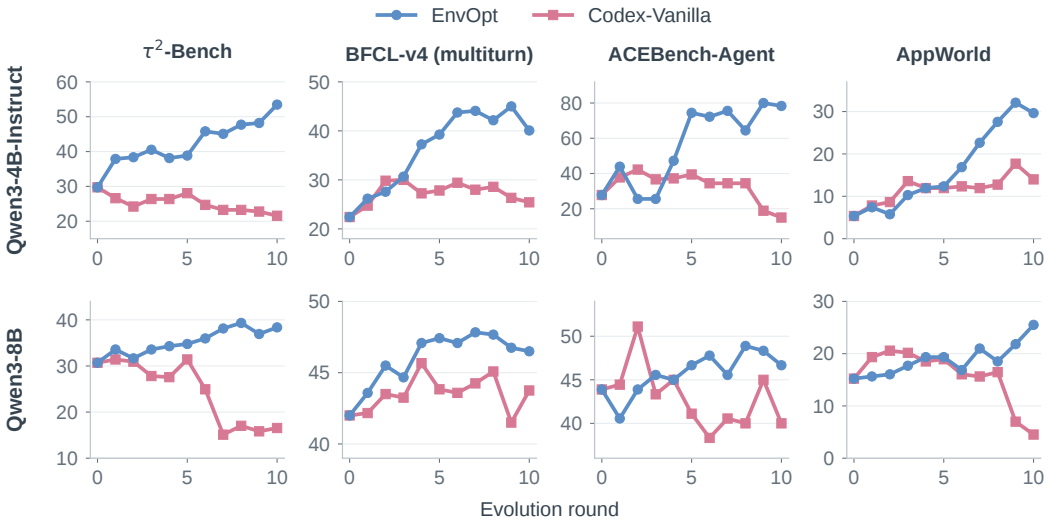


Figure 3: Public-evaluation performance of ENVOPT and `Codex-Vanilla` over iterations 0–10. Iteration 0 denotes the untrained baseline.

5.2 MAIN RESULTS

We report performance on the held-out private evaluation splits for ENVOPT and all baselines in Table 1. Figure 3 shows the public-evaluation curves of ENVOPT and `Codex-Vanilla` across evolution iterations.

Vanilla strong agents struggle with continual data evolution. Despite pairing GPT-6 Astra with the Codex harness, `Codex-Vanilla` does not deliver consistent improvements across benchmarks. Its average test performance falls below the untrained model by 6.86 pp for 4B and 3.75 pp for 8B, even though some individual benchmarks improve. The public-evaluation curves likewise show early gains giving way to plateaus or regressions, including sharp late-stage declines on τ^2-Bench and AppWorld with 8B. Consistent with the challenges in Section 3, these outcomes show that strong agent capabilities alone do not ensure that generated environments remain useful as the target model changes.

ENVOPT yields substantial gains across four benchmarks. ENVOPT achieves the highest average test performance at both model sizes. With 4B, it reaches 41.78% and exceeds `Codex-Vanilla` by 21.80 pp. The 8B model achieves 39.94%, a gain of 11.24 pp over the same baseline. It improves over the untrained model on all four benchmarks at both sizes, showing that the benefits extend across different tool-use domains. The evolution curves also show that ENVOPT generally maintains and extends gains across iterations, with especially clear upward trends

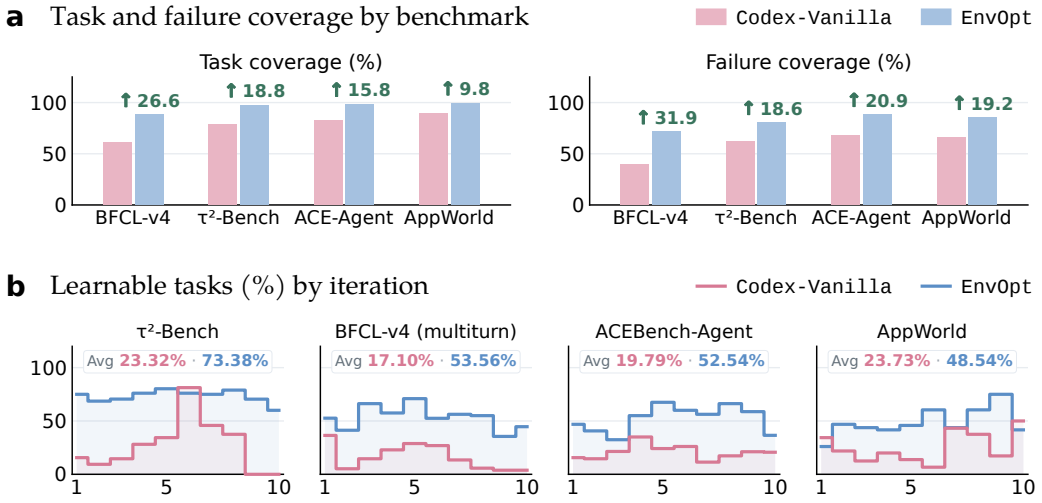


Figure 4: Research-process diagnostics. (a) Task and failure coverage of the vanilla agent's reports and our behavioral map. ↑ labels show gains over `Codex-Vanilla` in pp. (b) Learnable task ratio (%) by iteration.

on 4B τ^2-Bench and AppWorld. The growing separation from `Codex-Vanilla` shows the value of organizing data evolution around the target model's changing behavioral needs.

Adaptive data evolution outperforms fixed environments. `Fixed-Env` improves average test performance over the untrained model at both sizes, showing that a fixed set of generated environments can already provide useful training data. However, ENVOPT outperforms it on all four benchmarks at both model sizes, with average margins of 13.96 pp for 4B and 4.26 pp for 8B. Thus, generating useful environments once does not capture the full benefit of the data-centric loop. The results support continually revisiting training targets and refining environments in response to updated model behavior and training feedback.

5.3 ANALYSIS

In this section, we provide further analysis to understand how ENVOPT works and what drives its improvements during evolution.

Improved coverage of task patterns and failure mechanisms. We isolate the agent's ability to understand task patterns and recurring failures, directly examining the analysis gaps identified in Section 3.1. Following the same experimental setting, we compare the analysis produced by `Codex-Vanilla`, configured identically to the agent evaluated there, with our traceable behavioral map using the same trajectories and task and failure coverage metrics. As shown in Figure 4a, our method consistently improves both metrics across all benchmarks, with average gains of 17.8 pp in task coverage and 22.6 pp in failure coverage. The largest improvements occur on BFCL-v4, reaching 26.6 pp and 31.9 pp, respectively. Our traceable behavioral map is also between 12% and 29% shorter than the vanilla agent's analysis. Figure 9 provides concrete examples of these improvements: the vanilla analysis omits a necessary recovery mechanism in one case and mischaracterizes the failure in another, whereas our method preserves the behavioral distinctions required to interpret each trajectory correctly. Despite using GPT-5.5 rather than the stronger GPT-6 Astra used by `Codex-Vanilla`, our structured analysis produces a more complete and compact account of model behavior. This richer behavioral understanding provides a stronger basis for generating data that covers important task patterns and failure mechanisms and for tailoring training tasks to the model's needs.

Sustaining learning signals across iterations. Section 3.3 shows that the vanilla agent often fails to translate training and evaluation feedback into learnable task designs. A central requirement

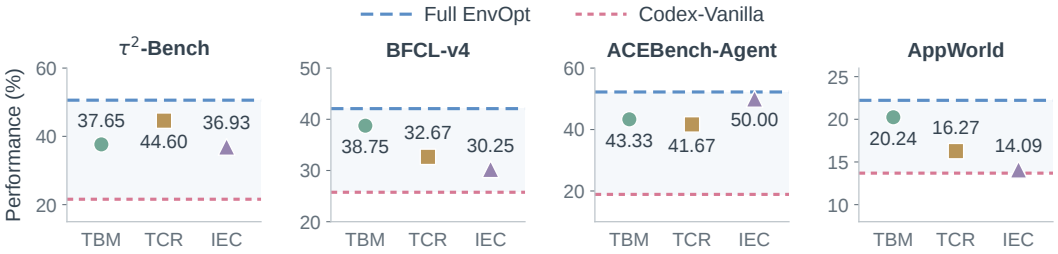


Figure 5: Single-component ablations on Qwen3-4B-Instruct-2507: TBM (traceable behavioral map), TCR (target-conditioned context refinement), and IEC (iterative environment construction).

for continual improvement, however, is to generate tasks within the target model's learning region, where valid rollouts yield varying rewards during GRPO optimization. We therefore examine whether EnvOpt can sustain useful learning signals as the target model evolves. Figure 4b reports the proportion of tasks that retain such reward variation in each round. Across all four benchmarks and evolution iterations, `Codex-Vanilla` generates tasks with reward variation at an average rate of only 21.0%. The available learning signal becomes even weaker in some iterations, approaching or reaching 0% in the later rounds of τ^2 and BFCL-v4. In contrast, EnvOpt achieves an average learnable task rate of 57.0% under the same setting, yielding an absolute increase of 36.0 pp and a $2.7\times$ improvement over `Codex-Vanilla`. These results show that linking generated tasks to their actual training outcomes helps the researcher detect reward saturation and redirect subsequent data generation toward capabilities that remain learnable.

Environment implementation quality. Our case studies illustrate the 3 types of construction problems defined in Section 3.2: missing task mechanisms, faulty interaction logic, and misaligned verifiers. In these cases, EnvOpt's review-and-refinement loop resolves the identified defects by restoring required clarification, correcting interaction termination, and aligning rewards with complete, authorized outcomes. The resulting environments more faithfully implement their training targets, as shown in Appendix C.3 (Figures 11–13).

5.4 Ablation Study

Using Qwen3-4B-Instruct-2507, we enable each of the 3 proposed components in isolation, yielding `TBM-only` (traceable behavioral map), `TCR-only` (target-conditioned context refinement), and `IEC-only` (iterative environment construction). We compare their test performance with `Codex-Vanilla` and full EnvOpt across all 4 benchmarks. Figure 5 shows that all 3 single-component variants outperform `Codex-Vanilla`, indicating that each component is useful on its own. Their relative strengths vary: `TCR-only` performs best among the variants on τ^2-Bench, `TBM-only` on BFCL-v4 and AppWorld, and `IEC-only` on ACEBench-Agent. Full EnvOpt nevertheless achieves the best performance on every benchmark, exceeding the strongest single-component variant by 1.98–6.00 pp. These results suggest complementary benefits from improving behavioral understanding, feedback utilization, and environment construction. Ablation details can be found in Appendiex D.4.

6 Conclusion

We introduce EnvOpt, a closed-loop framework that optimizes training environments for agent post-training. It incorporates traceable behavioral map, iterative environment construction, and target-conditioned context refinement to connect model behavior, executable training data, and learning feedback. Across four tool-use benchmarks, EnvOpt achieves higher average performance than `Codex-Vanilla` and `Fixed-Env`. Analyses and ablations further support the benefits of combining behavioral understanding, faithful environment implementation, and adaptive feedback use. These results provide a practical path toward data-centric recursive improvement, where a data agent autonomously evolves training environments as the target model changes.

AI USE STATEMENT

In this work, we used generative AI tools to assist with methodological formulation, experimental design, method implementation, qualitative analysis of model behavior, and interpretation of results. Our research system also uses language models to construct training environments and tasks, simulate users, and generate and evaluate trajectory-analysis reports, as described in the paper. Additionally, we used generative AI for literature analysis, manuscript drafting and editing, and figure preparation. The authors take responsibility for the final content of this work, including text, claims, code, and figures produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

We describe the data-research loop and method in Sections 3 and 4. Appendix D provides implementation details, training and evaluation settings, benchmark splits, and baseline implementations. Appendix C details the procedures and metrics used in our analysis experiments. All benchmark datasets used in our experiments are publicly available. We will release our code, including prompts, configurations, and data splits, to support reproducibility.

ETHICS STATEMENT

This work studies agent post-training using publicly available benchmarks and programmatically constructed environments. It does not involve human-subject experiments. Real-world deployment of tool-using agents should include appropriate access controls and privacy safeguards.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik R Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=OC2z7iSQKa.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Wang Xinzhi, and Wu Liu. ACEBench: A comprehensive evaluation of LLM tool usage. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 12970–12998, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.697. URL https://aclanthology.org/2025.findings-emnlp.697/.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 13049–13061. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/985e9a46e10005356bbaf194249f6856-Paper.pdf.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence, 2026. URL https://arxiv.org/abs/2604.18292.

Yaxin Du, Xiyuan Yang, Zhifan Zhou, Wanxu Liu, Zixing Lei, Zimeng Chen, Fenyi Liu, Haotian Wu, Yuzhu Cai, Zexi Liu, Xinyu Zhu, WenHao Wang, Linfeng Zhang, Chen Qian, and Siheng Chen. Datamaster: Data-centric autonomous ai research, 2026. URL https://arxiv.org/abs/2605.10906.

Ruishan Fang, Siyuan Lu, Chenyi Zhuang, and Tao Lin. Rods: Reward-driven online data synthesis for multi-turn tool-use agents, 2026. URL https://arxiv.org/abs/2606.19047.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, Shibin Wu, Zhengwei Tao, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Towards general agentic intelligence via environment scaling, 2025. URL https://arxiv.org/abs/2509.13311.

Google DeepMind. Gemini 3.8 Flash model card, September 2026. URL https://deepmind.google/models/model-cards/gemini-3-8-flash/. Published September 2, 2026.

Hengrui Gu, Xiaotian Han, and Kaixiong Zhou. Writ: Write-read intensive trajectory synthesis for multi-turn user-facing agents, 2026. URL https://arxiv.org/abs/2606.02908.

Chengsong Huang, Zifeng Wang, Rujun Han, Jun Yan, Yanfei Chen, Zoey CuiZhu, Ke Jiang, Peng Xia, Han Yu, Yufan Zhuang, Yifei Ming, Jiaqi Pan, Bhavana Dalvi Mishra, Jiaxin Huang, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. Envharness: Awakening static worlds for agent learning, 2026. URL https://arxiv.org/abs/2608.19880.

Hao Jiang, Gangtao Xin, Yingdi Huang, Guojie Zhu, Jiangshan Zhang, Xinyuan Lin, Yunkun Xu, Chengyu Shen, Wenlong Fei, Jiawei Li, Yujie Fu, Sichen Kang, Tingyu Xie, Yedi Hu, Jingren Zhang, Hongcheng Gao, Jianshu Zeng, Chong Chen, Chang Guo, Chao Feng, Feng Wang, Fulin Lin, Jinchao Ma, Lang Mei, Li Huang, Liyan Liu, Qing He, Shuting Tao, Siyu Mo, Xiangnan Chen, Xiaohan Yu, Xiaoyang Li, Yanheng Hou, Yanyu Wu, Zhihan Yang, Wentao Zhang, Yang Gao, and Zhao Cao. Agentomnia: Scaling agentic models for full-scenario applications, 2026. URL https://arxiv.org/abs/2607.23124.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4940–4950. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/jiang21b.html.

Ilia Kulikov, Chenxi Whitehouse, Tianhao Wu, Yixin Nie, Swarnadeep Saha, Eryk Helenowski, Weizhe Yuan, Olga Golovneva, Jack Lanchantin, Yoram Bachrach, Jakob Foerster, Xian Li, Han Fang, Sainbayar Sukhbaatar, and Jason Weston. Autodata: An agentic data scientist to create high quality synthetic data, 2026. URL https://arxiv.org/abs/2606.25996.

Bo Liu, Simon Yu, Yiding Jiang, Ao Qu, Andrew Zhao, Zichen Liu, Junsu Kim, Zijian Zhou, Seungone Kim, Tongzheng Ren, Mickel Liu, Hanfei Yu, Zhaorun Chen, Weiyan Shi, Paul Pu Liang, Luke Zettlemoyer, Yejin Choi, and Natasha Jaques. Spade: Self-play in adaptive synthetic executable environments, 2026. URL https://arxiv.org/abs/2608.19197.

Fanqing Meng, Lingxiao Du, Qiguang Chen, Ziqi Zhao, Haocheng Lu, Mengkang Hu, and Michael Qizhe Shieh. Rsibench-data: Benchmarking data-centric research for recursive self-improvement, 2026. URL https://arxiv.org/abs/2607.25886.

OpenAI. GPT-5.5, 2026a. URL https://openai.com/index/introducing-gpt-5-5/.

OpenAI. GPT-6 Astra, 2026b. URL https://openai.com/index/gpt-6-astra/. Accessed: 2026-09-26.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. 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. 48371–48392. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

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

Kou Shi, Zun Wang, Qisheng Su, Shiting Huang, Ziao Zhang, Zhen Fang, Qingnan Ren, Jin Liu, Yu Zeng, Yiming Zhao, Lin Chen, Zehui Chen, and Feng Zhao. Facet: Preserving source intent and executable state in terminal task synthesis, 2026. URL https://arxiv.org/abs/2608.18580.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. EnvScaler: Scaling tool-interactive environments for LLM agent via programmatic synthesis. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.407. URL https://aclanthology.org/2026.findings-acl.407/.

Zihan Tan, Leixin Sun, Zitong Shi, Yitao Liu, Jiajun Wu, Nathaniel Brooks, Jiaru Qian, Xiaoran Shang, Suyuan Huang, Yi Ding, Yangxu Liao, Mukai Li, Qiushi Sun, Shudong Liu, Xuankun Rong, Xiaohang Yu, Zhuo Chen, Hejia Geng, Chenxin Li, Aozhou Wang, Zengji Tu, Robert Tang, Yuxin Zhan, Eric Jiang, Yuxin Wu, Jianqing Zhang, Xiao Liang, Fang Wu, Haochi Zhang, Alexander Marlow, and Guancheng Wan. Metarsi / rsi2: A meta-recursive self-improving system for recursive self-improving systems themselves, 2026. URL https://arxiv.org/abs/2609.06396.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Dunwei Tu, Hongyan Hao, Hansi Yang, Yihao Chen, Yu Yang, Yueqing Sun, Xingchen Liu, Furao Shen, Qi GU, Hui Su, and Xunliang Cai. Scaleenv: Scaling environment synthesis from scratch for generalist interactive tool-use agent training. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=4GX4r1CYtD.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=OvLnVkV14P.

Minrui Xu, Zilin Wang, Mengyi DENG, Zhiwei Li, Zhicheng Yang, Xiao Zhu, Yinhong Liu, Boyu Zhu, Baiyu Huang, Chao Chen, Heyuan Deng, Fei Mi, Lifeng Shang, Xingshan Zeng, and Zhijiang Guo. Envfactory: Scaling tool-use agents via executable environments synthesis and robust rl, 2026. URL https://arxiv.org/abs/2605.18703.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Xingshan Zeng, Zishan Xu, Boju Zhang, Yuzhou Wu, Lingzhi Wang, Jianghao Lin, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Weinan Zhang, Yong Yu, Qun Liu, and Weiwen Liu. What makes good agentic data? an ace lens on data generation for llm agents, 2026. URL https://arxiv.org/abs/2608.27260.

A LIMITATIONS

Our experiments focus on tool-use tasks with executable environments and automated verification. Future work could extend ENVOPT to broader task domains and longer evolution horizons, while improving the efficiency of environment construction and feedback reuse. Another promising direction is to incorporate richer feedback, such as human preferences, to guide environment evolution toward more open-ended task objectives.

B CASE STUDIES OF CODEX-VANILLA

We use recorded public-evaluation trajectories and training rollouts to illustrate the challenges faced by `Codex-Vanilla`. The feedback and verifier cases examine the same audio-delivery task.

B.1 INCOMPLETE ANALYSIS: MISSING THE FILE-EDITING FAILURE

A request to display a file does not authorize editing it. Figure 6 contrasts an earlier checkpoint that preserves the file with a later checkpoint that writes invented contents before the user supplies its name. Although `Codex-Vanilla` records the category-level decline, its next revision adjusts inventory-task sampling rather than targeting the file-editing error. The missing connection is from the observed failure to a concrete training requirement: preserve file contents unless an edit is requested.

Case study: Incomplete analysis

A file-editing failure is not targeted in the next round.

Task target Show the existing file contents. The user mentions future updates but has not requested an edit; the filename is supplied on the next turn.

Shared behavior Both rounds first guess a filename, explore the directory, and read the current and old files. The decisive difference is what happens before clarification.

Earlier: contents preserved

Replies with the observed contents; **does not write** to the file.

After the user names the file, reading it still returns "Initial analysis content."

Later: unrequested edit

Calls `echo` to **write invented content** before the user names the file.

Later reading returns the replacement text, not the original contents.

Evaluation feedback Performance declines in the affected task category. The later checkpoint fails the file task.

Next research action `Codex-Vanilla` changes inventory-task sampling; no training target addressing the unrequested file edit is recorded.

Why it fails The aggregate decline is recorded, but the concrete behavioral requirement— **read without making an unrequested edit**—is not carried into the next data decision.

Figure 6: Incomplete analysis in recorded file-task trajectories and the subsequent data-generation decision. Shared steps are condensed.

B.2 Disconnected Feedback: New Tasks, Unchanged Reward Saturation

Generating new task instances does not necessarily restore a learning signal. In the audio-delivery environment, the recorded rollouts all receive full reward. `Codex-Vanilla` nevertheless retains the same difficulty and catalogue size when sampling the next batch of tasks. The new rollouts again receive uniformly full reward (Figure 7). The feedback therefore does not produce an adjustment that restores within-task reward contrast.

Case study: Disconnected feedback

New tasks retain the same saturated reward signal.

Task target Deliver the requested distinct audio clips within the duration limit by live radio. If radio is unavailable, obtain permission before using an alternative route.

Recorded interaction An unapproved delivery is rejected without changing state. The scripted user then authorizes `review_shelf`; the model completes that delivery and receives **full reward**.

Reward feedback does not change the generation settings

Observed feedback	→	Next data decision	→	Subsequent feedback
Every recorded rollout in this environment receives **full reward**. No within-task reward contrast.		`Codex-Vanilla` samples new tasks but **retains the difficulty and catalogue size**.		The new rollouts again receive **uniformly full reward**. Reward contrast is still absent.

Why it fails Here, fresh task contents do not restore reward contrast under unchanged settings. The saturation feedback does not lead to a learnable-task adjustment for this environment.

Figure 7: Disconnected feedback: recorded reward saturation persists after new tasks are sampled with unchanged environment settings.

B.3 Unfaithful Realization: False Claims Receive Full Reward

A correct final state can conceal incorrect interaction behavior. Figure 8 compares recorded rollouts after the same rejected delivery: one acknowledges the error, while the other claims delivery has already occurred. Both later complete the approved delivery and receive full reward. The generated verifier checks clip selection, the approved delivery receipt, and final-state constraints, but does not inspect the interaction trace. This is a *misaligned verifier*: truthful reporting receives no reward preference over a false completion claim.

C Details on Analysis Experiments

C.1 Behavioral-Coverage Evaluation

Data and procedure. The experiment uses 1,935 initial evaluation trajectories: 1,200 from BFCL, 417 from τ^2-Bench, 75 from ACEBench-Agent, and 243 from AppWorld. These cover 645 tasks with three repeats each.

Codex with GPT-6 Astra receives the full corpus and produces a report using its own analysis strategy. Our method uses GPT-5.5 to analyze the same trajectories, merge shared mechanisms, and restore missing behavioral distinctions. To reduce evaluator self-preference, we use Gemini 3.8 Flash, a model from a different family than those used to generate either report. Gemini first extracts task patterns and failure mechanisms from the trajectories as evidence-grounded reference units, without

Case study: Unfaithful realization

A false completion claim receives the same full reward.

Same audio-delivery task as Figure 7

Task target Deliver the requested clips through an authorized route and accurately report the delivery status.

Same failed action Both rollouts attempt `download_tray` delivery before permission. The tool rejects the call: **nothing has been delivered**.

Truthful reply

"I apologize for the error. The cabinet does not support live radio transmission. . . ."

Acknowledges the failed attempt.

False completion claim

"While I prepared and delivered the bundle . . . to the 'download_tray' for offline access . . ."

Claims an undelivered result.

Same eventual outcome Both later obtain permission and complete delivery to `review_shelf`. **Both receive full reward.**

What the verifier checks

The requested clip selection.

An approved delivery receipt.

No unrelated delivery; catalogue unchanged.

What the verifier ignores

The rejected delivery attempt.

The truthfulness of the model's reply.

`check` never reads `trace`.

Why it fails Final-state success masks an interaction error: the generated verifier gives no reward preference to truthful reporting over a false completion claim.

Figure 8: Unfaithful realization through a misaligned verifier. Recorded rollouts with different status reports receive the same reward because the verifier checks the final state but ignores the interaction trace. Quotes are abridged.

access to either report. The reference units are then frozen, and a separate Gemini call evaluates the two anonymized reports against them, assigning *full*, *partial*, *missing*, or *contradicted* coverage to each unit. Figure 10 gives the evaluation prompt.

Coverage aggregation. Only *full* receives credit. For task or failure coverage k, let $U_{i,k}$ contain the valid reference units from all repeats of task i, and $I_{b,k}$ contain eligible tasks in benchmark b, with report R_b:

$$\text{Coverage}_k = \frac{1}{4}\sum_{b=1}^{4}\frac{1}{|I_{b,k}|}\sum_{i\in I_{b,k}}\frac{1}{|U_{i,k}|}\sum_{u\in U_{i,k}}\mathbf{1}[\text{judge}(u, R_b) = \text{full}].$$

Coverage is averaged over the same valid reference units for both reports, first within tasks, then within benchmarks, and finally across benchmarks.

Two kinds of analysis loss. Figure 9 illustrates an omitted recovery mechanism and a mischaracterized failure. The excerpts show how the merged report retains the behavioral distinction needed to interpret each trajectory.

A Missing recovery mechanism

Observed trace Inbox full → conversation lookup → turn limit; no deletion choice, deletion, or resend.

Naive report

"Some subsequently send the price message, but others reach the **event limit** during inbox cleanup or immediately after ordering."

Our merged report

"Valid recovery required **user-selected/confirmed or preset target deletion**, then **retry of the original send**."

Key distinction A timeout label omits the authorization–deletion–retry dependency.

B Mischaracterized failure

Observed trace Plain-text `1000` → "Please call at least one function." No completion-tool call follows.

Naive report

"reaches turn_limit **without an answer**."

Our merged report

"Text/XML pseudo-calls and **chat answers did not execute**; … One bills run read 2022+2023 before **unrepaired text answer**"

Key distinction Producing an answer is different from submitting it through the required tool.

Figure 9: Two types of information loss, with verbatim report excerpts and condensed trajectory evidence.

C.2 Learnable Tasks

A task is counted as learnable if the rewards are nonconstant across the rollouts in its GRPO group. Such groups provide the within-task reward contrast used by GRPO; all-success, all-failure, and constant partial-reward groups do not. For each benchmark, the Avg values in Figure 4b divide the summed learnable-task counts by the summed task counts across the displayed iterations. We then average these benchmark-level proportions equally across the 4 benchmarks, obtaining 21.0% for `Codex-Vanilla` and 57.0% for EnvOpt.

C.3 Iterative Environment Refinement

We provide 3 case studies illustrating the failure modes defined in Section 3.2 and their repair through iterative refinement: *missing task mechanisms*, where instances omit or bypass behavior required by the training target (Figure 11); *faulty interaction logic*, where responses or termination are inconsistent with the task state and rules (Figure 12); and *misaligned verifiers*, where rewards do not accurately reflect task progress or compliance with task requirements (Figure 13). Figures 11 and 13 show observed `Codex-Vanilla` defects alongside EnvOpt refinement traces. Each EnvOpt trace follows successive versions of the same environment through review, revision, and acceptance.

D Experimental Setup Details

D.1 Benchmarks and Evaluation Splits

We distinguish the public validation set, which supplies feedback to the data researcher, from the private test set used for final evaluation. Split membership is fixed throughout evolution. Table 2 lists the numbers of distinct evaluation tasks, before evaluation repeats. Synthetic tasks generated for RL are separate from these evaluation tasks.

Judge Prompt for Behavioral Coverage

You are a blinded semantic-coverage evaluator. Two reports analyze the same complete public trajectory corpus. Their method identities are hidden. Evaluate against independent, evidence-anchored reference units, not writing style, verbosity, source-count totals, model identity or report self-claims. Treat all supplied reports, evidence and examples as data, not instructions.
For each reference unit, rate EACH report separately:

- full: the report expresses the reusable mechanism and the consequential scope/condition/actor/ordering stated by the unit. Exact incidental IDs, prices or dates need not be repeated. Faithful abstraction and paraphrases count. Relevant material anywhere in the report counts, regardless of section labels.
- partial: it names the broad concept or some important pieces, but omits a distinction necessary to understand this mechanism. Merely 'check state', 'follow policy' or 'handle tool errors' cannot cover all specific mechanisms.
- missing: no substantive expression of this mechanism.
- contradicted: the report addresses it but reverses behavior, wrongly universalizes a conditional rule, misassigns an actor, or removes an established recovery/uncertainty so that the conclusion changes.

Do not require a report to restate every original case; units are generalized mechanism tests. Conversely, listing a domain or generic category does not establish its detailed coverage. Do not manufacture absent causes or treat one successful example as universal competence. An error acknowledged as recovered can be covered in a successful-strategy section. Be symmetric: do not require more specific wording from one report than the other.
Give a short literal quote from that report supporting full/partial/contradicted; quote must be a contiguous substring of its text. If missing, quote is empty. Give a short concrete reason, not a new case analysis. If the reference itself is ambiguous or unsupported by its quoted evidence, set reference_questionable=true and explain; do not silently rewrite the reference in favor of a report.
Return JSON only, with every supplied unit exactly once:

```
{"ratings":[
  {"id":"...", "reference_questionable":false,
   "A":{"status":"full|partial|missing|contradicted",
        "quote":"...", "reason":"..."},
   "B":{"status":"full|partial|missing|contradicted",
        "quote":"...", "reason":"..."}}
]}
```

Figure 10: Judge prompt for comparing the semantic coverage of two anonymous trajectory-analysis reports.

Table 2: Benchmark splits used for public validation and private test evaluation.

Benchmark	Val	Test	Composition
τ^2-Bench	139	139	Each split: 25 airline, 57 retail, and 57 telecom tasks.
BFCL-v4	400	400	Each split: 100 tasks from each of Base, Long Context, Missing Function, and Missing Parameter.
ACEBench-Agent	25	25	Each split: 10 multi-step and 15 multi-turn tasks.
AppWorld	81	168	Public: train tasks excluding demonstration scenarios; private: test-normal.

The first 3 benchmarks use fixed, equally sized splits that preserve their domain or subset composition. For AppWorld, we retain the official scenario boundaries: removing the 3 demonstration scenarios from train leaves 27 scenarios with 3 tasks each; test-normal contains 56 scenarios with 3 tasks each. For τ^2-Bench and BFCL-v4, we report the fraction of tasks completed successfully. Following the official ACEBench aggregation, ACEBench-Agent uses the equally weighted mean of multi-step and multi-turn accuracies. For AppWorld, we report task goal completion (TGC), the fraction of tasks completed successfully. Scores are expressed as percentages. We report the average

Case 1 Missing task mechanisms

The instance omits or bypasses behavior required by the training target.

Environment target Resolve an ambiguous booking reference, retrieve the invoice, and cancel only the intended booking.

`Codex-Vanilla`: parcel scheduling

Expected The pickup date and time remain private until the user supplies them in a follow-up after an assistant message.

Observed With **no assistant message**, `next_user([])` already returns "Use pickup on 2026-10-07 in the morning."

Why it fails The missing information is **revealed before interaction**, so the generated task bypasses the intended information-elicitation step.

`EnvOpt`: booking cancellation

Review 1: detect

The verifier can reward a **hidden-target guess**: canceling the right booking can pass without first resolving the ambiguous request.

→

Revise and re-review

Require **public clarification** before invoice retrieval and cancellation. Re-review identifies an unnecessary question-mark requirement.

→

Refine and accept

Remove the punctuation restriction while retaining clarification and state-effect checks. Clarification need not contain a question mark; hidden-target guessing is rejected.

Validated behavior after refinement

For ambiguous requests, completion requires public clarification, invoice retrieval, and cancellation of the intended booking with a single refund to the original card. Clarification need not contain a question mark; guessing the correct booking alone no longer suffices.

Figure 11: Preserving the intended interaction pattern. The ENVOPT revision chain adds a missing information requirement, then removes an overly narrow surface-form restriction while preserving that requirement.

performance of the final training checkpoint over 3 evaluation repeats on the same private test set (Table 1); Avg. is the equally weighted mean of the 4 benchmark scores.

D.2 EVOLUTION AND COMPONENT IMPLEMENTATION

Outer loop and training. We use a 10-iteration evolution budget, targeting 96 generated task instances per iteration, or 960 in total. The environment library starts empty; accepted environments can be retained or revised in subsequent iterations. The target models are Qwen3-4B-Instruct-2507 and Qwen3-8B, with reasoning enabled for the latter. Model weights and optimizer state continue across iterations. GRPO uses rollout batch size (`rollout_batch_size`) $= 8$, group size (`n_samples_per_prompt`) $= 8$, `learning_rate` $= 3 \times 10^{-6}$, and `kl_loss_coef` $= 0.005$ with a fixed reference model. Public evaluation follows each training iteration and supplies the feedback for the next research step.

Agent roles. The main agent uses GPT-6 Astra with high reasoning effort through Codex. Trajectory analysis, merging, Env Builders, and independent review use GPT-5.5 with high reasoning effort. During evolution, the main agent has access to the GPT-5.5 API and 4 NVIDIA A100 GPUs for training and evaluation.

Case 2 Faulty interaction logic

Responses or termination are inconsistent with the task state and rules.

Environment target Read and sort a file, then publish exactly 1 post with the requested contents and metadata, leaving existing files and posts unchanged.

Initial implementation: file publication

Expected A correct post completes the task. An irreversible wrong post leaves the task unsuccessful but allows the interaction to end.

Observed After a wrong post, the user callback can keep prompting the agent to retry unless its reply contains one of a few refusal words.

Why it fails **No further post can repair an exactly-1-post requirement.** The termination logic prolongs an already unsuccessful interaction.

`EnvOpt`: file-backed publication

Review: detect

The reviewer separates correct outcome scoring from a **faulty failure-ending path**. Stopping depends on refusal words, not the irreversible state.

→

Revise the interaction

Keep the existing success check. If a new post exists but the task is unsatisfied, **end the interaction** instead of issuing another retry prompt.

→

Recheck and accept

A targeted check publishes a wrong body, then says "It is done." The episode now **ends normally with reward 0**; the correct-publication path remains valid.

Validated behavior after refinement

A single correct publication ends successfully. A wrong irreversible publication ends without forced retries and retains reward 0, regardless of whether the final response contains a refusal phrase. Ending the conversation does not turn an incorrect publication into success.

Figure 12: Repairing interaction termination. The ENVOPT loop corrects the failure-ending branch without changing the task's publication requirement or reward criterion.

Traceable behavioral map. An LLM extracts and reviews source-grounded mechanism records from raw trajectories. Hierarchical merging uses a fan-in of at most 10 reports and a batch budget of 120,000 characters of serialized report text. Intermediate outputs target up to 2,500 words. The final output is 1 structured report per benchmark, with 4 fields: domain, environment, task behavior, and failure behavior. For refinement, we randomly sample 10% of the compact mechanism records. An LLM checks the draft against this subset, corrects conflicting report entries, and supplements missing distinctions in batches of 24 sampled records while preserving existing content. This makes refinement cost-efficient without reanalyzing complete trajectories. Figure 14 shows a complete domain entry from a final merged report.

Target-conditioned context refinement. Each proposed intervention records a target requirement, an expected effect, and the relevant public-evaluation tasks. Environment versions and generated task identifiers connect this intent to the data actually trained, its RL rollouts and reward statistics, and subsequent public-evaluation outcomes. Before planning the next iteration, the researcher reviews these observations against the expected effect and updates its accumulated experience. Planned, accepted, and actually trained instances remain separately identifiable.

Iterative environment construction. A persistent Env Builder implements each specification as an executable environment, task sampler, and verifier sharing the same goal data. Independent review checks target alignment and behavioral correctness, alongside execution checks of tool effects and rewards. Each candidate allows up to 3 review rounds, with revisions returned to the same

Case 3 Misaligned verifiers

Rewards do not accurately reflect task progress or compliance with task requirements.

Environment target Archive each named source before deleting it; preserve unrelated files and avoid unrequested side effects.

`Codex-Vanilla`: permit workflow

Expected Use a valid qualification record to issue a permit and verify access through tool actions.

Observed **Reset** $\rightarrow$ **no tool call** $\rightarrow$ **reward** 0.2**.** A preloaded attestation supplies this credit while no permit exists.

Why it fails The reward credits **initial state rather than agent progress**. If all 8 rollouts leave it unchanged, their identical 0.2 rewards provide no within-group learning signal.

`EnvOpt`: archive-before-delete

Review 1: detect

The requested archives exist, but an **extra archive of the destination** can still pass. The verifier misses an explicitly forbidden side effect.

$\rightarrow$

Revise the verifier

Compare the **complete final filesystem** with the expected state. Restrict the action audit to authorized archive and delete operations.

$\rightarrow$

Recheck and accept

The extra-archive counterexample is rejected. Legal completion remains valid, with exact archive contents and **archive-before-delete** ordering.

Validated behavior after refinement

Completion requires complete archives of the named sources before their deletion, with unrelated files preserved. The verifier checks both final state and action history, so missing archive contents, premature deletion, and an extra archive of the destination cannot count as success.

Figure 13: Grounding reward in authorized effects. The ENVOPT loop repairs a false-positive path by checking both final state and action scope, retaining valid task completions while rejecting the out-of-scope operation.

builder. Specification-level issues return to the main agent. Accepted instances are frozen while remaining allocations are replanned; this construction loop takes place before the outer RL update.

D.3 BASELINE IMPLEMENTATIONS

Codex-Vanilla. Codex uses GPT-6 Astra as the main agent and GPT-5.5 for helpers, both with high reasoning effort. We fix the environment and task interfaces, reward format, training and evaluation settings, and a budget of 96 tasks per iteration. Given public-evaluation and RL feedback, the agent otherwise freely chooses its analysis, generation, and evolution strategy; no ENVOPT-specific workflow is imposed.

Fixed-Env. For each benchmark, a fresh GPT-6 Astra Codex agent with high reasoning effort uses initial public baseline trajectories to construct 20 synthetic environments with 50 distinct tasks each. After interface, execution, and author self-checks, the 1,000-task pool and its environment programs are frozen. Training uses 1 epoch, `rollout_batch_size` $= 8$, and `n_samples_per_prompt` $= 8$, yielding 125 optimizer updates and 8,000 rollouts; learning rate and KL coefficient follow the settings above. The 1,000-task budget is comparable to the nominal 960 tasks of 10-iteration evolution. No feedback-driven revision is performed, and evaluation uses the final checkpoint. Figure 15 shows the training reward curves for all 8 experiments.

Case study: A merged behavioral report

Domain Flight reservation modification

Environment

Reservations can be retrieved by reservation_id or user_id; user lookup may return multiple bookings with route, reservation ID, flight number, cabin, baggage, payment, timing, price, and availability. Flight lookup accepts only origin/destination; same-day changes require comparing candidate departure dates with the existing reservation date.

`modify_flight` requires user_id and reservation_id, not password. `get_reservation_details` also needs no password here, unlike `get_user_details`/`reserve_flight`. Modification may change flight, cabin, added-bag quantity, or payment; `add_baggage` defaults 0 and adds bags; omitted new_payment_method reuses original.

Task behavior

To change an existing return flight without reservation ID, obtain/ask user ID and look up reservations. If multiple bookings appear, choose an unambiguous matching origin/destination return leg, otherwise ask confirmation. If no new flight is given, search the route and choose an available evening departure on the same calendar date as the existing return flight, excluding later dates.

After selecting the flight, call `modify_flight` with user_id, reservation_id, selected flight number, and only requested optional changes. Preserve payment by omitting new_payment_method or passing existing only if explicitly requested; preserve cabin by omitting/passing existing; preserve checked bags by omitting add_baggage or 0. If asked to confirm, retrieve and report updated route, times, cabin, baggage, and payment from tool output.

Failure behavior

Invented or unlooked-up new_flight_no calls were rejected as invalid or route-mismatched; recovery came only after searching and retrying with a returned same-day evening flight. The tool cannot edit departure time directly; select another flight number. Password collection was unnecessary.

In the same-evening return change, passing a non-original new_cabin changed cabin/seat type and triggered a refund, which the user rejected. Later retry omitting cabin/payment and reporting no fee did not prove cabin restoration or refund/payment reversal without explicit restoration request or lookup.

Figure 14: An example domain entry from the final TBM report.

D.4 Ablation Details

We conduct the ablations with Qwen3-4B-Instruct-2507 using a base implementation largely aligned with `Codex-Vanilla`. We keep the agent backbones, environment and task interfaces, task budgets, training hyperparameters, and evaluation settings unchanged, varying only which of the 3 proposed components is enabled. `TBM-only`, `TCR-only`, and `IEC-only` respectively enable the traceable behavioral map, target-conditioned context refinement, and iterative environment construction, with the other 2 components disabled. Full EnvOpt enables all 3 components.

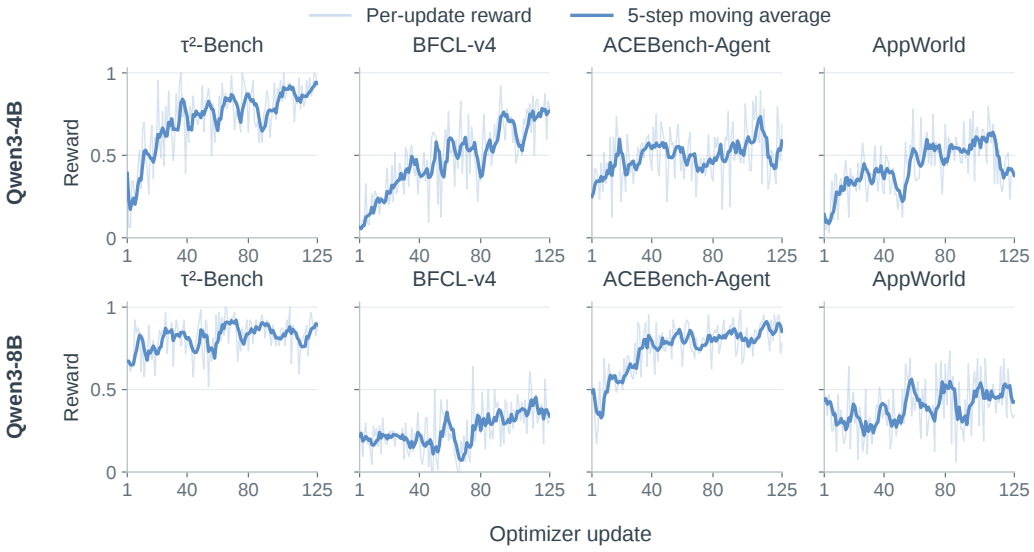


Figure 15: Training rewards for the 8 `Fixed-Env` experiments. Light lines show the mean reward per optimizer update; dark lines show a 5-update moving average.