ENVIRONMENT EVOLUTION FOR TERMINAL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaling interactive and verifiable environments is critical for training terminal agents. As frontier models become more capable, environments synthesized from scratch become less challenging and thus provide limited learning signals. Recent co-evolution methods iteratively synthesize environments near the model's learnable frontier based on weaknesses exposed during rollouts. However, their dependence on on-policy rollouts limits generalization and the continuous provision of learning signals as the model becomes stronger. In this paper, we propose *environment evolution*, which incrementally increases environment difficulty off-policy and schedules the evolved environments generation by generation during training to provide continuous learning signals. We derive three evolution directions that influence environment difficulty from the multi-turn learning objective and then implement evolution along these directions through a loop-engineered multi-agent harness. Quantitative rollout experiments with Hy4 preview, Claude Opus 5, and GPT-5.6 Sol show that environment evolution consistently produces more difficult environments. We validate its effectiveness on Qwen3.6-27B and Qwen3.6-35B-A3B through simple long-horizon RL training, improving their performance by 14.4 and 18.0 percentage points on Terminal-Bench 2.1, respectively.

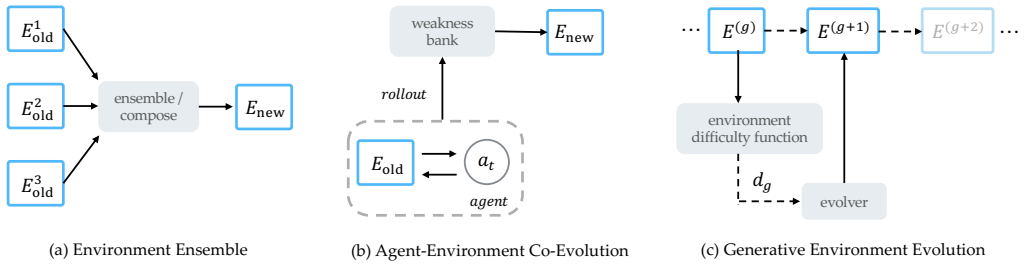
1 INTRODUCTION

Reinforcement learning environments are emerging as the next scalable direction for training capable agents (Bellemare et al., 2013; Brockman et al., 2016). With industrial-scale agentic RL algorithms stabilizing and asynchronous infrastructure maturing (Fu et al., 2026; Cao et al., 2025; Tan et al., 2025), the focus of further scaling is shifting toward the environments. For general-purpose terminal agents, the difficulty and diversity of environments are therefore the central levers that determine what agents can learn through verifiable feedback: adaptive difficulty preserves continual learning potential, while diversity supports generality (Dennis et al., 2021; Jiang et al., 2021; Parker-Holder et al., 2023; Garcin et al., 2024; Cobbe et al., 2020; Team et al., 2021; Merrill et al., 2026).

Recent work has focused on synthesizing large-scale terminal environments from scratch through human-designed pipelines that transform diverse resources (e.g., GitHub repositories, skills, and webpages) into executable terminal environments, each with a corresponding instruction (what to do) and a verification system (how completion is assessed) (Gandhi et al., 2026; Wu et al., 2026; Fan et al., 2026; Pi et al., 2026; Hua et al., 2026; Zhao et al., 2026; Yao et al., 2026). However, these environments are often insufficiently challenging for current frontier models, which consistently solve them across repeated rollouts. When used for RL training, such environments are discarded because they fail to provide effective learning signals that distinguish between better and worse trajectories, wasting environment construction costs and reducing diversity in the retained training distribution.

To provide useful learning signals, existing co-evolution methods couple model training with environment synthesis by using on-policy rollouts on seed environments to expose weaknesses that guide the synthesis of new environments that remain challenging yet learnable (Zala et al., 2024; Hu et al., 2025; Guo et al., 2025; Sygkounas et al., 2026). However, the resulting environments are constrained by the rollout model and initial environment distribution, limiting generalization and their ability to continuously provide learning signals as training saturates and model failures become sparse.

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.



(a) Environment Ensemble (b) Agent-Environment Co-Evolution (c) Generative Environment Evolution

Figure 1: Comparison of paradigms for environment scaling. **(a)** Environment ensemble obtains a new environment by composing several difficult primitive environments. **(b)** Agent–environment co-evolution rolls out the target agent on primitive environments to populate a weakness bank, which guides the construction of new environments. **(c)** Environment evolution, our proposed paradigm, evolves the environment itself: an evolver uses a derived difficulty signal to produce successive environments along the lineage.

In this paper, we propose *environment evolution*, which increases environment difficulty generation by generation without relying on rollout models, as shown in Figure 1.

From the multi-turn learning objective, we derive an off-policy formulation of environment difficulty, which identifies *scenario* novelty, *skill* rarity, and execution *length* as three factors that influence difficulty. We then implement environment evolution with a loop-engineered multi-agent harness that incrementally modifies an existing environment along one selected from these three directions, which constructs lineages of increasing difficulty while introducing diverse variations around the original environment distribution. Rollout-based difficulty estimates with Hy4 preview, Claude Opus 5, and GPT-5.6 Sol show that the evolved environments are challenging across different models despite being synthesised off-policy. During model development, we find evolved environments continue to provide challenging RL signals even when their seed environments have already been used for SFT.

Experiments on Qwen3.6-27B and Qwen3.6-35B-A3B demonstrate that environment evolution improves Terminal-Bench 2.1 performance by 14.4 and 18.0 percentage points, respectively. Compared with agent environment co-evolution, off-policy environment evolution provides longer-lasting learning signals and achieves better performance. In summary, our contributions are as follows:

1. We derive a model-agnostic formulation of environment difficulty from the multi-turn learning objective, showing that environment evolution provides a more general approach to increasing environment difficulty.

2. We propose *environment evolution* and implement it as a loop-engineered multi-agent harness that incrementally evolves existing environments to construct verified lineages of increasing difficulty.

3. We validate the approach through 200-step long-horizon RL experiments on Qwen3.6-27B and Qwen3.6-35B-A3B, demonstrating longer-lasting learning signals than co-evolution and ensemble baselines and improvements of 14.4 and 18.0 percentage points on Terminal-Bench 2.1, respectively.

2 Related Work

Terminal agents. Terminal agents interact with computing systems through command-line tools (Chen et al., 2021; Yang et al., 2024; Merrill et al., 2026), granting them open-ended access to explore and exploit computational resources and iteratively use environment feedback to complete long-horizon tasks (Yao et al., 2023). A line of work on harness design aims to strengthen agents' planning, navigation, and exploration while constraining undesirable behaviors, with a focus on observation-space design, context compression, and tool routing (Lee et al., 2026; Wang et al., 2025; Bui, 2026). The community has recently turned to synthesizing large-scale terminal environments from scratch for agent training (Gandhi et al., 2026; Wu et al., 2026; Fan et al., 2026; Pi et al.,

2026; Hua et al., 2026; Zhao et al., 2026; Yao et al., 2026), while RST (Li et al., 2026) focuses on extending reference solutions to evolve environments. Despite their abundance and broad domain coverage, our large-scale rollout and quality assessment experiments reveal that these open-source environments suffer from low-quality reward signals (e.g., misalignment between task instructions and verification systems, corrupted environments) (Bercovich, 2026) and are insufficiently challenging to provide meaningful learning signals for frontier models.

Environment Scaling. Open-ended reinforcement learning requires a continual stream of solvable yet challenging environments that retain learning potential (Wang et al., 2019; Dennis et al., 2021; Jiang et al., 2021; Parker-Holder et al., 2023). Paired Open-Ended Trailblazer (POET) co-evolves a population of environment–agent pairs, generating new challenges through environment mutation and transferring agents across environments to exploit stepping stones (Wang et al., 2019; 2020). Unsupervised Environment Design (UED) formalizes the automatic construction and curation of valid and solvable environments from underspecified environment parameters, encompassing regret-based generation, prioritized replay, and incremental level editing (Dennis et al., 2021; Jiang et al., 2021; 2022; Parker-Holder et al., 2023). Recent work has begun to bring these ideas to LLM agents: through feedback-conditioned generation (Chen et al., 2026; Yang et al., 2026), online curricula (Qi et al., 2025), and agent–environment co-evolution (Guo et al., 2025; Liu et al., 2026), these methods adapt the environment distribution as the agent improves, keeping environments near its capability frontier. However, they require a designated agent to estimate environment difficulty through on-policy rollouts, and the resulting environments are related to both the rollout agent and the initial environment distribution. Instead, environment evolution constructs increasingly difficult lineages independently of the target policy and schedules successive generations during training to provide continual learning signals.

3 PRELIMINARIES

Instead of estimating difficulty on-policy, e.g., by rolling out a model and using its pass rate as the difficulty metric, we need an off-policy metric that measures the difficulty of the environment itself, derived from the multi-turn learning objective. Since models are trained on different data distributions, a difficulty estimate tied to one model θ is model-specific weakness rather than environment difficulty.

We first view agent-environment interaction as a Markov process (Kaelbling et al., 1998) with interleaved observations and actions. Let $h_t = (o_{\le t}, a_{<t})$. Then $o_t \sim O_{\mathcal{E}}(\cdot \mid s_t)$, $a_t \sim \pi_\theta(\cdot \mid h_t, g)$, and $s_{t+1} \sim P_{\mathcal{E}}(\cdot \mid s_t, a_t)$, which induces a low-level execution trajectory $\zeta = (o_0, a_0, o_1, a_1, \ldots, o_T)$.

Following prior definitions from hierarchical agent execution (Sutton et al., 1999), we treat agent execution trajectory at a higher level as an interleaving of scenarios and skill executions:

$$\xi = (\sigma_0, \kappa_1, \sigma_1, \ldots, \kappa_L, \sigma_L),$$

where σ_t is the high-level scenario at step t, and κ_t is the skill applied under that scenario.

Under model θ, the likelihood of a high-level trajectory decomposes over the scenarios it reaches and the skills it applies. Taking the negative log-likelihood gives the model-specific difficulty:

$$\begin{aligned} D_\theta(\xi) &= -\log p_\theta(\xi \mid g) \\ &= \sum_{t=1}^{L} \Big(-\log p_\theta(\sigma_{t-1} \mid g) - \log p_\theta(\kappa_t \mid \sigma_{t-1}, g) \Big). \end{aligned}$$

This quantity has three contributors. First, L is the number of meaningful solver turns required by the trajectory. Second, $-\log p_\theta(\sigma_{t-1} \mid g)$ measures scenario novelty under the model. Third, $-\log p_\theta(\kappa_t \mid \sigma_{t-1}, g)$ measures the rarity of applying the required skill in that scenario. The last two terms are policy-dependent: they depend on the model's training-data distribution and learned policy.

To obtain a policy-independent difficulty measure, we replace the model-dependent probabilities with those under a reference distribution $\mathcal{T}$ grounded in broad world knowledge:

$$D_{\mathcal{T}}(\xi) = \sum_{t=1}^{L} \Big(-\log p_{\mathcal{T}}(\sigma_{t-1} \mid g) - \log p_{\mathcal{T}}(\kappa_t \mid \sigma_{t-1}, g) \Big).$$

Here $p_{\mathcal{T}}(\sigma \mid g)$ measures how common a scenario is within the environment family, and $p_{\mathcal{T}}(\kappa \mid \sigma, g)$ measures how common the required skill is under that scenario. This converts the estimate from model-specific weakness into model-agnostic environment difficulty. In particular, a deep-research agent can estimate both distributions through broad web search, grounding them in world knowledge, by providing the relevant context of g and ξ.

This also gives a direct way to relate environment difficulty to agent weakness. Let $z_t = (\sigma_{t-1}, \kappa_t)$ denote the scenario-skill requirement at step t, and define the per-step difficulties

$$d_\theta(z_t \mid g) = -\log p_\theta(\sigma_{t-1} \mid g) - \log p_\theta(\kappa_t \mid \sigma_{t-1}, g),$$

and $d_{\mathcal{T}}(z_t \mid g)$ analogously under $p_{\mathcal{T}}$. Agent weakness is the excess difficulty that remains after subtracting the environment-family difficulty:

$$\delta_\theta(z_t \mid g) = \left[d_\theta(z_t \mid g) - d_{\mathcal{T}}(z_t \mid g)\right]_+ .$$

where $[x]_+ = \max(x, 0)$. Equivalently,

$$\delta_\theta(z_t \mid g) = \Big[\log \frac{p_{\mathcal{T}}(\sigma_{t-1} \mid g)}{p_\theta(\sigma_{t-1} \mid g)} + \log \frac{p_{\mathcal{T}}(\kappa_t \mid \sigma_{t-1}, g)}{p_\theta(\kappa_t \mid \sigma_{t-1}, g)}\Big]_+ .$$

For the full high-level trajectory,

$$\Delta_\theta(\xi) = \sum_{t=1}^{L} \left[d_\theta(z_t \mid g) - d_{\mathcal{T}}(z_t \mid g)\right]_+ .$$

Thus, weakness is not the difficulty of the environment itself; it is the portion of that trajectory that is unusually difficult for a particular model relative to the environment family.

This distinction clarifies the scope of on-policy co-evolution of agent and environment. Such methods collect rollouts from a model θ, identify the model's failure modes, and generate new environments around those failures. If e_t^θ indicates a failure at step t, the induced signal is mainly $\sum_{t=1}^{L} e_t^\theta[-\log p_\theta(\kappa_t \mid \sigma_{t-1}, g)]$. That is, on-policy co-evolution of agent and environment primarily targets skill-selection errors under scenarios contained in the seed environments and reached by the current model during rollouts. It does not explicitly control the number of required solver steps L, nor does it systematically increase scenario novelty under $p_{\mathcal{T}}(\sigma \mid g)$. In contrast, the environment evolution directly operates on the full difficulty space:

$$(L,\ -\log p_{\mathcal{T}}(\sigma \mid g),\ -\log p_{\mathcal{T}}(\kappa \mid \sigma, g))$$

which shows that environment evolution offers a more general paradigm for providing continuous learning signals.

For a detailed derivation and discussion of the underlying assumptions, please refer to Appendix D.

4 Approach

4.1 Sequence-Guided Environment Evolution.

Environment evolution is implemented as a loop-engineered multi-agent harness, as illustrated in Figure 2. It takes the latest accepted environment as input and produces the next-generation environment through an incremental modification guided by the expected execution trajectory at the scenario and skill levels.

Loop 1: Plan Refinement The Proposer first extracts an execution sequence of interleaved scenarios and skills from E:

$$\xi_E = (\sigma_1, \kappa_1), \ldots, (\sigma_L, \kappa_L). \tag{1}$$

It then updates the sequence according to the evolution direction selected at the current generation. For *length*, it inserts scenario–skill pairs into the sequence to introduce additional dependencies along the expected execution trajectory. For *scenario*, it replaces one scenario while preserving its paired skill; for *skill*, it replaces one skill. A plan is generated from the difference between the updated and original sequences, and a rubric-based reviewer iteratively reviews it until it is accepted or the current evolution direction fails.

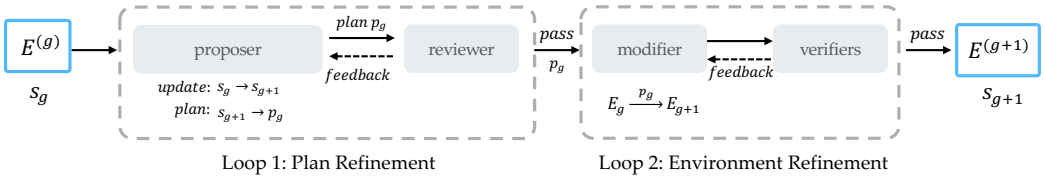


Figure 2: Loop-engineered multi-agent harness for environment evolution. It decomposes each generation into two gated feedback loops: (1) sequence-guided plan refinement, which generates and revises an evolution plan until it passes rubric-based review; and (2) plan-conditioned environment refinement, which evolves and repairs the candidate until strict solvability and quality checks pass.

Loop 2: Environment Refinement The reviewed plan is then passed to the Modifier, which creates a residual ΔE and applies it to the current environment. Since each modification is incremental, the environment distribution can gradually shift along the model's learning frontier toward the real-world distribution. Each candidate must pass three verifiers run in parallel: (i) an Oracle verifier checks that the reference solution succeeds in the sandbox; (ii) an Invalid-test verifier confirms that an empty or no-op solution fails, ensuring that the verification system is reliable; and (iii) an adaptive general-rubrics verifier checks environment quality, where rubrics are aligned with the full set of official Terminal-Bench criteria for contributed tasks. During development, accepted environments undergo human-in-the-loop review. Issues that bypass the rubric-based checks are converted into new rubrics for the plan reviewer and environment verifier until the loop reliably produces environments with no issues identified by human reviewers.

Evolution effort. To control the mutation between adjacent generations, we introduce a prompt-controlled mutation parameter called *evolution effort*, analogous in spirit to thinking effort, with three levels: low, high, and max. While the evolution direction determines what type of sequence-level edit is performed, evolution effort controls its scope. The three effort levels restrict the edit to one pair $(\sigma_\ell, \kappa_\ell)$, one contiguous span, or an unrestricted portion of the sequence, respectively. Section 5 quantitatively validates the effectiveness of this design.

At each generation, we randomly order the three evolution directions. If the Plan Reviewer rejects a plan or the repair budget for the current direction is exhausted, we fall back to the next direction, construct a new target sequence from the same environment, and repeat the process. The branch terminates only when all three directions fail.

4.2 Evolution-Lineage Scheduler.

As evolution proceeds, later generations become increasingly difficult. Sampling randomly from the full lineage can therefore expose the policy to environments it cannot yet solve, yielding all-failure rollout groups that provide no effective learning signal. We therefore propose the Evolution-Lineage (EL) Scheduler, which starts from the earliest generation and schedules its environments in order. Let $E_{i,g,k}$ be the k-th environment in generation g of lineage i, with $N_{i,g}$ environments in that generation. At update u, the scheduler updates the active indices as follows:

$$(g_{u+1}, k_{u+1}) = \begin{cases} (g_u, k_u), & \widehat{p}_u(E_{i,g_u,k_u}) \le \tau, \\ (g_u, k_u + 1), & \widehat{p}_u(E_{i,g_u,k_u}) > \tau \ \wedge \ k_u < N_{i,g_u}, \\ (g_u + 1, 1), & \widehat{p}_u(E_{i,g_u,k_u}) > \tau \ \wedge \ k_u = N_{i,g_u}, \end{cases} \qquad \widehat{p}_u(E) = \frac{1}{B} \sum_{b=1}^{B} r_{u,b}. \tag{2}$$

We use $B = 8$ and $\tau = 6/8$. Once the current environment exceeds this threshold, the scheduler moves to the next environment in the same generation. It advances to the next generation only when no environments remain in the current one.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Harness. In this paper, evaluation and RL training use the Claude Code harness (Anthropic, 2025). It fixes the tool protocol across models while improving execution efficiency by running multiple tool calls in parallel within a single assistant turn. For RL training, the harness operates with a 256K context window and automatically compacts the trajectory when the remaining usable context reaches 16K, providing stable context management for long-horizon execution.

Benchmark. Terminal-Bench 2.1 Verified (Merrill et al., 2026) is the primary held-out benchmark for terminal agent capability. It fixes instabilities in Terminal-Bench 2.0 that hinder reproducible evaluation and incorrectly underestimate benchmark performance. Each task is configured with 32 CPU cores and 48 GB of memory, with a timeout of 4 hours. Sampling uses temperature 1.0, top-p 0.95, and top-k 20, with a dynamic output budget, $256\text{K} - L_{\text{used}}$, at each assistant turn, to avoid truncation errors caused by an overly long turn. We report the average over five runs.

Training Algorithm. We use GRPO (Shao et al., 2024) for agentic RL training with partial rollouts, fully asynchronous GPU location, and a staleness bound of 5 to reduce GPU bubble time. When auto compact of Claude Code harness is triggered, its summary is retained in the complete trajectory and treated as a regular action turn for multi-turn credit assignment. We train Qwen3.6-27B, a 27B-parameter dense model, and Qwen3.6-35B-A3B, a mixture-of-experts model with 35B total and 3B activated parameters. Both checkpoints provide a native context length of 262,144 tokens. For the MoE model, we additionally use R3 (Ma et al., 2025) to stabilize training process.

Monitoring Metrics. **Environment Difficulty** is estimated by the pass rate over 8 independent rollouts. We also record the average number of assistant turns as a measure of long-horizon execution; assistant turns typically account for nearly half of the total turns. Rollouts use Claude Opus 5 and GPT-5.6 Sol with `xhigh` effort and Hy4 preview (Tencent Hunyuan, 2026) with `high` effort, all with a 1M context window. **Environment Mutation.** For each generation, structural change relative to the seed is measured across instruction, environment, and verification system, with their mutations measured at the token, file, and test-unit levels, respectively.

Seed Environment Selection. We collect 47,678 non-benchmark terminal environments from Hugging Face and GitHub and retain 127 through strict rubric-based filtering for environment quality and solvability. Each candidate must include an executable Oracle solution that passes the verifier, satisfy rubric-based quality checks (we find environment quality critical to successful RL training), and meet the difficulty threshold under Claude Opus 5: a pass rate of at most $4/8$ and average turns of at least 30. SkillSynth (Fan et al., 2026) is then used to supplement the pool with newly synthesized environments, which are subject to the same quality and difficulty filters. Finally, uniform sampling across domains yields a balanced and diverse seed pool of 500 environments.

5.2 EVOLUTION EFFORT

Starting from the same seed environments, the randomized cross-mode strategy independently constructs 15-generation lineages under `low`, `high`, and `max` evolution effort. We use a common cap of 15 generations because pass rate provides no further resolution once a lineage enters the zero-pass regime; beyond that point, the effectiveness of additional evolution cannot be assessed reliably from rollout outcomes.

As shown in Figure 3, `low` effort progressively extends long horizon execution, as avg turns increase across generations, but its pass rate fluctuates and remains above zero. Because `low` modifies only one local pair $(\sigma_\ell, \kappa_\ell)$, successive generations may repeatedly edit the same pair and partially return to an earlier configuration. In contrast, `high` and `max` monotonically reduce pass rate to zero and increase avg turns, with `max` reaching zero earlier and producing the larger change in difficulty.

Balancing effective and stable evolution against the controllability of each generation, `high` is used as the default evolution effort. Appendix E reports the corresponding mutation rates for instruction, environment, and verification system components.

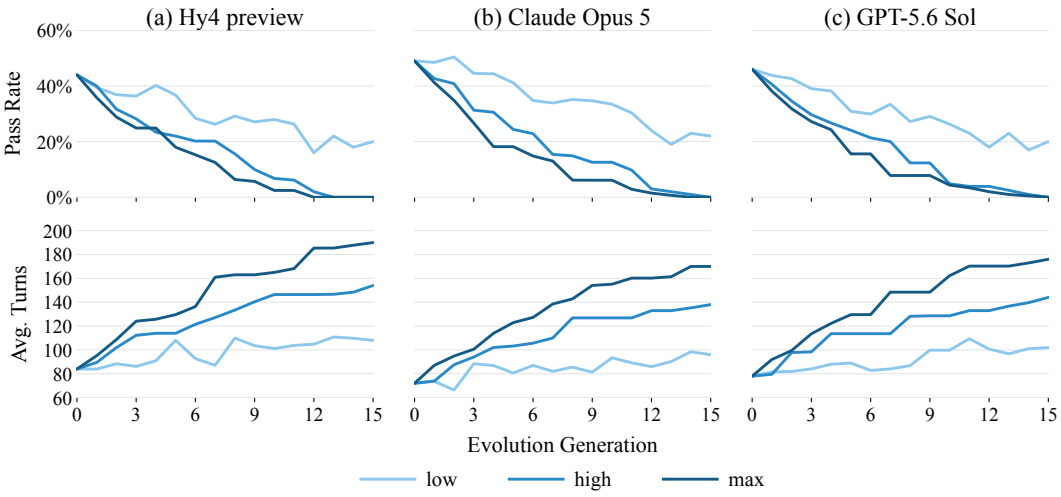


Figure 3: Environment difficulty across 15 evolution generations under `low`, `high`, and `max` evolution effort, evaluated independently by Hy4 preview, Claude Opus 5, and GPT-5.6 Sol.

Table 1: Single-step consistency of evolution-direction effects under `high` evolution effort. The two difficulty-change columns are measured by Claude Opus 5 with `xhigh` thinking effort. The first block applies each direction once to the same seed environments. The second evolves each direction for 15 consecutive steps and averages the metrics over the 15 generation transitions. Agreement between the two blocks tests whether each direction preserves a stable, direction-specific change profile along longer lineages. Difficulty changes are computed as the later generation minus the previous generation, component columns report generation-level mutation rates, and Total is their unweighted mean.

Direction	Difficulty Change		Component Mutation (%)			
	Δ Pass Rate	Δ Avg Turns	Instruction	Environment	Verification	Total
			1-step effect			
scenario	−4.7 pp	+13.5	99.8%	59.6%	53.8%	71.1%
skill	−4.0 pp	+12.5	99.1%	50.3%	58.4%	69.3%
length	−7.1 pp	+9.4	87.5%	45.2%	58.6%	63.8%
			15-step mean effect			
scenario	−2.9 pp	+9.5	97.1%	34.6%	43.7%	58.5%
skill	−2.7 pp	+10.6	96.5%	27.8%	45.8%	56.7%
length	−4.8 pp	+7.4	85.5%	34.3%	48.9%	56.2%

5.3 Evolution Direction

To isolate the effect of evolution direction, we fix `high` evolution effort and apply `scenario`, `skill`, and `length` to the same seed environments. At each generation, we measure changes in pass rate and avg turns together with instruction, environment, and verification system mutation. The 1-step effect captures the immediate change induced by each direction, while the 15-step mean averages the same metrics over adjacent transitions in a direction-specific lineage to test whether that effect persists.

All three directions consistently reduce pass rate and increase avg turns. `length` produces the largest pass-rate decrease, whereas `scenario` and `skill` produce larger increases in avg turns. `scenario` yields the largest total mutation, while `length` achieves the strongest pass-rate reduction with the smallest total mutation. The agreement between the 1-step and 15-step results indicates that these direction-specific profiles persist beyond a single edit. In the full evolution procedure, the

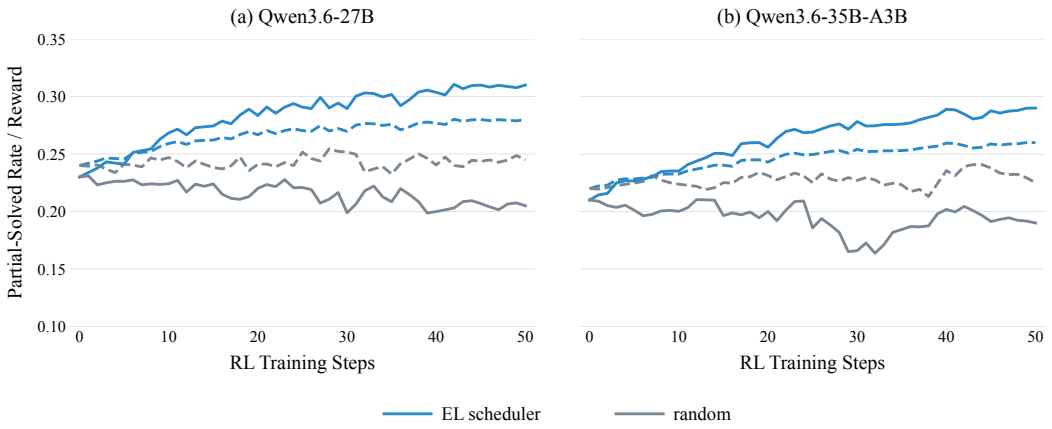


Figure 4: Early training dynamics over the first 50 training steps. Solid curves show the fraction of rollout groups that are partially solved, with one to seven successful trajectories among eight rollouts; dashed curves show mean training reward.

directions are therefore randomly ordered at each generation to diversify the resulting lineages, with cross-mode fallback whenever the current direction fails.

5.4 RL Training Dynamics

Before RL training, each base model generates trajectories for rejection sampling fine-tuning (RFT) (Touvron et al., 2023). Only trajectories that pass the verification system and conform to the Claude Code protocol are retained. The filtered set is rebalanced for trajectory diversity in order to increase policy entropy for better RL training. Starting from the resulting RFT checkpoint, denoted step 0, we train each model with GRPO for 200 steps.

EL Scheduler. The EL Scheduler advances through each environment lineage in order, exposing generation $g + 1$ only after generation g reaches the pass-rate threshold in Equation 2. Figure 4 compares it with random scheduling under matched rollout budgets during the first 50 RL steps. By avoiding premature exposure to generations that the policy cannot yet solve, the scheduler uses the rollout budget more efficiently and provides GRPO with more informative learning signals, as more rollout groups remain partially solved and retain non-zero within-group advantage. The EL Scheduler is used by default throughout the full 200-step RL training run.

Turns and Tokens. As the scheduler admits harder environments, turns and tokens per trajectory increase during training; Appendix E provides the detailed curves and analysis.

5.5 Comparison

Direct RL on Existing Environments. Training directly on the 500 selected seed environments yields no statistically significant improvement on either Qwen3.6-27B or Qwen3.6-35B-A3B at the 95% confidence level. We also apply minimal difficulty filtering to the collected environments, such as removing those solved in every rollout, and use all remaining environments for RL, but still observe no improvement. The lack of improvement from direct RL is consistent with the limited gains reported by TMax (Ivison et al., 2026) on a stronger starting model. While TMax attributes the limited gains to large-scale post-training of an already strong model, we attribute our lack of improvement to the insufficient difficulty and quality of the training environments.

For the following three environment-scaling methods, Claude Opus 5 is fixed as the environment synthesis model, and the total number of training environments is held constant to ensure a fair comparison.

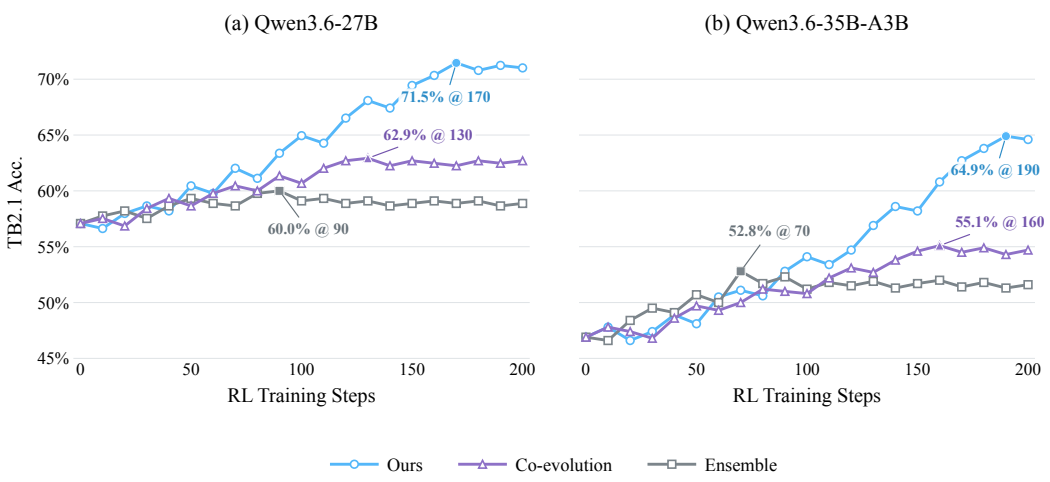


Figure 5: Checkpoint performance for Qwen3.6-27B (left) and Qwen3.6-35B-A3B (right) over 200 RL training steps, with offline evaluation every 10 steps; annotations mark each method's peak accuracy and corresponding training step. Step 0 denotes the shared checkpoint after RFT used to increase policy entropy before RL training.

Environment Ensemble. The target agent is first evaluated on the seed environments, and the environments it fails to solve are selected. All two- and three-environment combinations are then enumerated, with each combination merged into one harder environment. This procedure is applied recursively as a tree, with the outputs of one level becoming the inputs to the next.

Agent–Environment Co-Evolution. The target agent first rolls out on the seed environments to generate trajectories. Its failed trajectories are analyzed to construct a weakness bank, from which combinations are sampled to synthesize new, harder environments. After the agent trains for 50 RL steps, the procedure is repeated.

Environment Evolution. Environment Evolution randomly orders the three modes at each generation and follows the cross-mode fallback, producing a verified 15-generation lineage from each seed without target-agent rollouts.

Figure 5 reports offline Terminal-Bench 2.1 evaluation every 10 RL steps. Environment evolution reaches peak accuracies of 71.5% and 64.9% on Qwen3.6-27B and Qwen3.6-35B-A3B, compared with 62.9% and 55.1% for Co-evolution and 60.0% and 52.8% for Ensemble.

6 Conclusion

In this paper, we propose *environment evolution*, which incrementally synthesizes environments of increasing difficulty for RL training. It serves as a more general paradigm for scaling environments for terminal agents, built on the derived off-policy formulation of environment difficulty. Across different models, evolved environments show a consistent increase in difficulty generation by generation despite being synthesized off-policy. By scheduling these environments along lineages, long-horizon RL training on Qwen3.6-27B and Qwen3.6-35B-A3B receives continuously effective learning signals and achieves higher performance than co-evolution and ensemble baselines. Future work will explore the potential of environment evolution for SWE agents and Computer-Use Agents.

AI USE STATEMENT

Large language models are used for environment synthesis and verification as described in the paper. We also used AI tools to assist with translation, manuscript editing, checking and elaborating mathematical derivations, and LaTeX preparation. The authors are responsible for the final content and claims.

ETHICS STATEMENT

This work studies terminal-agent training with automatically generated executable environments. Automatically generated tasks and solutions can exploit gaps in task-specific verifiers. Verification scores should therefore be interpreted within the assumptions and coverage of each verifier, and solutions intended for downstream use should undergo independent validation. The computational costs of environment synthesis and reinforcement learning should also be considered when assessing the practical benefits of the method.

REPRODUCIBILITY STATEMENT

Experimental settings and data filtering are described in Section 5, and derivations and assumptions are detailed in Appendix D. All seed environments used in this work are publicly available. We will publicly release the prompts and evolved environments upon acceptance.

REFERENCES

Anthropic. Claude Code. https://code.claude.com/docs/en/overview, 2025.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents, 2013. URL https://arxiv.org/abs/1207.4708.

Ivan Bercovich. What makes a good terminal-agent benchmark task: A guideline for adversarial, difficult, and legible evaluation design, 2026. URL https://arxiv.org/abs/2604.28093.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540.

Nghi D. Q. Bui. Building effective ai coding agents for the terminal: Scaffolding, harness, context engineering, and lessons learned, 2026. URL https://arxiv.org/abs/2603.05344.

Shiyi Cao, Dacheng Li, Fangzhou Zhao, Shuo Yuan, Sumanth R. Hegde, Connor Chen, Charlie Ruan, Tyler Griggs, Shu Liu, Eric Tang, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Skyrl-agent: Efficient rl training for multi-turn llm agent, 2025. URL https://arxiv.org/abs/2511.16108.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Sara Cao, Yuanhao Xiong, Haibo Tong, Huaxiu Yao, Hengduo Li, Jiacheng Zhu, Xian Li, Dawn Song, Bo Li, Jason Weston, and Dat Huynh. Scaling agent learning via experience synthesis. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=cf7qpBwttr.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning, 2020. URL https://arxiv.org/abs/1912.01588.

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, 2021. URL https://arxiv.org/abs/2012.02096.

Zhiyuan Fan, Tinghao Yu, Yuanjun Cai, Jiangtao Guan, Yun Yang, Dingxin Hu, Jiang Zhou, Xing Wu, Zhuo Han, Feng Zhang, and Lilin Wang. Toward scalable terminal task synthesis via skill graphs, 2026. URL https://arxiv.org/abs/2604.25727.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. Areal: A large-scale asynchronous reinforcement learning system for language reasoning, 2026. URL https://arxiv.org/abs/2505.24298.

Kanishk Gandhi, Shivam Garg, Noah D. Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling rl environments for terminal agents, 2026. URL https://arxiv.org/abs/2601.16443.

Samuel Garcin, James Doran, Shangmin Guo, Christopher G. Lucas, and Stefano V. Albrecht. Dred: Zero-shot transfer in reinforcement learning via data-regularised environment design, 2024. URL https://arxiv.org/abs/2402.03479.

Jiacheng Guo, Ling Yang, Peter Chen, Qixin Xiao, Yinjie Wang, Xinzhe Juan, Jiahao Qiu, Ke Shen, and Mengdi Wang. Genenv: Difficulty-aligned co-evolution between llm agents and environment simulators, 2025. URL https://arxiv.org/abs/2512.19682.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jianguang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation, 2025. URL https://arxiv.org/abs/2408.00764.

Zhanbo Hua, Yifan Yao, Weihao Xie, Yongchi Zhao, Minghao Liu, Ruizhi Qiu, Zhewei Huang, Zun Wang, Yiyan Ji, Yunhai Ye, Letian Zhu, Xinping Lei, Han Li, Zhiyuan Ma, Zili Wang, Zhaoxiang Zhang, and Jiaheng Liu. CLI-Universe: Towards verifiable task synthesis engine for terminal agents, 2026. URL https://arxiv.org/abs/2606.22883.

Hamish Ivison, Junjie Oscar Yin, Rulin Shao, Teng Xiao, Nathan Lambert, and Hannaneh Hajishirzi. TMax: A simple recipe for terminal agents, 2026. URL https://arxiv.org/abs/2606.23321.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay, 2021. URL https://arxiv.org/abs/2010.03934.

Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design, 2022. URL https://arxiv.org/abs/2110.02439.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses, 2026. URL https://arxiv.org/abs/2603.28052.

Zhongzhi Li, Yucheng Shi, Zongxia Li, Ruhan Wang, Anhao Li, Zixun Huang, Junyao Yang, Lei Ke, Ninghao Liu, Haitao Mi, and Leowei Liang. Recursive synthesis for long-horizon terminal tasks, 2026. URL https://arxiv.org/abs/2608.05466.

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.

Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Zhifang Sui, and Fuli Luo. Stabilizing MoE reinforcement learning by aligning training and inference routers, 2025. URL https://arxiv.org/abs/2510.11370.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design, 2023. URL https://arxiv.org/abs/2203.01302.

Renjie Pi, Grace Lam, Mohammad Shoeybi, Pooya Jannaty, Bryan Catanzaro, and Wei Ping. On data engineering for scaling llm terminal capabilities, 2026. URL https://arxiv.org/abs/2602.21193.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c66e1fcc9691aae706250638f36f681b-Abstract-Conference.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.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1.

Alkis Sygkounas, Rishi Hazra, Andreas Persson, Pedro Zuidberg Dos Martires, and Amy Loutfi. Covolve: Adversarial co-evolution of large-language-model-generated policies and environments via two-player zero-sum game, 2026. URL https://arxiv.org/abs/2603.28386.

Zheyue Tan, Mustapha Abdullahi, Tuo Shi, Huining Yuan, Zelai Xu, Chao Yu, Boxun Li, and Bo Zhao. Earl: Efficient agentic reinforcement learning systems for large language models, 2025. URL https://arxiv.org/abs/2510.05943.

Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, Nat McAleese, Nathalie Bradley-Schmieg, Nathaniel Wong, Nicolas Porcel, Roberta Raileanu, Steph Hughes-Fitt, Valentin Dalibard, and Wojciech Marian Czarnecki. Open-ended learning leads to generally capable agents, 2021. URL https://arxiv.org/abs/2107.12808.

Tencent Hunyuan. Hy4 preview. https://hy.tencent.com/research/hy4-preview, August 2026.

Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions, 2019. URL https://arxiv.org/abs/1901.01753.

Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeff Clune, and Kenneth O. Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions, 2020. URL https://arxiv.org/abs/2003.08536.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2025. URL https://arxiv.org/abs/2407.16741.

Siwei Wu, Yizhi Li, Yuyang Song, Wei Zhang, Yang Wang, Riza Batista-Navarro, Xian Yang, Mingjie Tang, Bryan Dai, Jian Yang, and Chenghua Lin. Large-scale terminal agentic trajectory generation from dockerized environments, 2026. URL https://arxiv.org/abs/2602.01244.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. URL https://arxiv.org/abs/2405.15793.

Shidong Yang, Ziyu Ma, Tongwen Huang, Yiming Hu, Yong Wang, and Xiangxiang Chu. CoEvolve: Training LLM agents via agent-data mutual evolution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pp. 23015–23036, 2026. doi: 10.18653/v1/2026.acl-long.1055. URL https://aclanthology.org/2026.acl-long.1055/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Yihang Yao, Bo Pang, Xuan Phi Nguyen, Ding Zhao, Shafiq Joty, and Semih Yavuz. Learning generalizable behaviors for terminal agents, 2026. URL https://arxiv.org/abs/2608.22631.

Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. Envgen: Generating and adapting environments via llms for training embodied agents, 2024. URL https://arxiv.org/abs/2403.12014.

Jiarong Zhao, Zhikai Lei, Zhiheng Xi, Rui Zheng, Hang Yan, Jie Zhou, Qin Chen, and Liang He. NexForge: Scaling agent capabilities through requirement-driven task synthesis for LLMs, 2026. URL https://arxiv.org/abs/2607.14186.

A Version Note

We will provide more details in future versions. Future versions will also report RSI results in which the same model both constructs and learns from the environments.

B Further Scaling and Continual Improvement

We extend RL training from 200 to 500 steps but do not observe continued performance gains. To investigate this plateau, we analyze the resulting trajectories and decompose individual Terminal-Bench 2.1 tasks into multiple simpler question-answering probes to evaluate the trained models. These analyses suggest that the remaining bottleneck is a lack of the knowledge required to solve these tasks.

We attribute this limitation to the different roles of the training stages. The relevant knowledge is primarily acquired from the pretraining corpus and through controlled knowledge injection during mid-training. Post-training, especially reinforcement learning, primarily teaches models to use their existing knowledge and acquire new skills through exploration. However, acquiring missing knowledge directly through exploration remains difficult.

We therefore interpret these results as indicating that continual improvement in model performance requires coordinated progress across multiple training stages, rather than scaling reinforcement learning alone.

C Computational Cost

Across all experiments, we incurred approximately USD 150,000 in API costs. The six main RL experiments consumed a total of 129,024 GPU-hours of training.

D Derivation and Interpretation of Environment Difficulty

This section details the connection to the multi-turn learning objective and the modeling assumptions behind the difficulty formulation in Section 3.

Learning objective and trajectory likelihood. Let g denote the task goal and context, and let $r_E(\zeta) \in \{0, 1\}$ indicate whether a trajectory completes the task according to the environment's verifier. For discrete trajectories, the task-success objective is

$$J_E(\theta; g) = \mathbb{E}_{\zeta \sim P_{\theta,E}(\cdot|g)}[r_E(\zeta)] = \sum_{\zeta \in \mathcal{S}_E(g)} P_{\theta,E}(\zeta \mid g), \tag{3}$$

where $\mathcal{S}_E(g)$ is the set of successful trajectories and $P_{\theta,E}$ is the interaction distribution induced by the policy and environment. For $J_E(\theta; g) > 0$, maximizing task success is equivalent to minimizing $-\log J_E(\theta; g)$. Any successful reference trajectory $\zeta^\star$ with positive probability satisfies

$$-\log J_E(\theta; g) \leq -\log P_{\theta,E}(\zeta^\star \mid g). \tag{4}$$

This motivates trajectory surprisal as a difficulty proxy associated with a selected successful execution. Its value depends on that execution: a task with many valid solutions can be easy even when any individual trajectory has low probability. The bound concerns the exact trajectory likelihood; the factorization below introduces an additional modeling approximation.

Exact high-level factorization. Let $H_t = (\sigma_{0:t-1}, \kappa_{1:t-1})$ denote the scenario and skill histories before κ_t, with an empty skill history at $t = 1$. Conditional on a fixed high-level length L, the chain rule gives

$$\begin{aligned} P_{\theta,E}(\xi \mid g, L) = P_{\theta,E}(\sigma_0 \mid g, L) \\ \times \prod_{t=1}^{L} P_{\theta,E}(\kappa_t \mid H_t, g, L) P_{\theta,E}(\sigma_t \mid H_t, \kappa_t, g, L). \end{aligned} \tag{5}$$

This expression includes the scenario transitions and terminal scenario σ_L. Low-level Markov dynamics alone do not make the high-level scenario–skill requirements independent.

Factorized requirement model. To expose the factors that environment evolution can modify, represent an execution plan by $z_{1:L}$, where $z_t = (\sigma_{t-1}, \kappa_t)$ pairs each required skill with its preceding scenario. We adopt the factorized model

$$q_\theta(z_{1:L} \mid g, L) = \prod_{t=1}^{L} p_\theta(\sigma_{t-1} \mid g) p_\theta(\kappa_t \mid \sigma_{t-1}, g), \tag{6}$$

with normalized distributions over a fixed abstraction of scenarios and skills, and positive probabilities for the requirements being scored. This approximation summarizes each requirement by its local scenario–skill pair and scores the required execution steps rather than the terminal outcome. The compact likelihood notation $p_\theta(\xi \mid g)$ in Section 3 refers to this factorized requirement model, with the conditioning on L left implicit. Taking its negative logarithm yields

$$\begin{aligned} D_\theta(\xi) &:= -\log q_\theta(z_{1:L} \mid g, L) \\ &= \sum_{t=1}^{L} \left[-\log p_\theta(\sigma_{t-1} \mid g) - \log p_\theta(\kappa_t \mid \sigma_{t-1}, g) \right]. \end{aligned} \tag{7}$$

This separates execution length, scenario novelty, and skill rarity. Holding the existing requirements fixed, adding a requirement with positive surprisal increases the total score. Here L counts high-level execution steps; one skill execution may involve multiple assistant turns, so L need not equal the observed *avg turns*. The Approach section indexes scenarios by requirement pair: its pair $(\sigma_\ell, \kappa_\ell)$ corresponds to $(\sigma_{\ell-1}, \kappa_\ell)$ under the state-boundary indexing used here.

World-knowledge reference. Replacing the model-dependent requirement distributions with a fixed reference $p_\mathcal{T}$ gives $D_\mathcal{T}$ by the same decomposition. The reference describes the relative prevalence of scenarios and skills in the relevant world-knowledge context. It remains fixed with respect to the target policy, making the resulting measure independent of that policy while still relative to the chosen reference and abstraction. The formulation is used to identify and guide the three evolution directions; it does not require explicitly computing normalized probabilities or evaluating $D_\mathcal{T}$ numerically. A deep-research agent can ground relative judgments of scenario novelty and skill rarity in world knowledge using the task and execution context, without numerically fitting the full reference distribution.

Model-specific weakness. Under the factorized model, the difference between model-dependent and reference difficulty is

$$d_\theta(z_t \mid g) - d_\mathcal{T}(z_t \mid g) = \log \frac{p_\mathcal{T}(\sigma_{t-1} \mid g)\, p_\mathcal{T}(\kappa_t \mid \sigma_{t-1}, g)}{p_\theta(\sigma_{t-1} \mid g)\, p_\theta(\kappa_t \mid \sigma_{t-1}, g)}. \tag{8}$$

Model-dependent difficulty therefore includes both reference difficulty and a model-relative deviation. We define weakness as the positive part of this deviation, $\delta_\theta(z_t \mid g) = [d_\theta(z_t \mid g) - d_\mathcal{T}(z_t \mid g)]_+$. Summing these positive parts gives $\Delta_\theta(\xi)$, so strengths on some requirements do not cancel weaknesses on others.

Scope of the co-evolution comparison. The failure-weighted skill term in Section 3 is an illustrative description of feedback from a failure-conditioned weakness bank, rather than a general objective for all co-evolution methods. Failure-driven updates can also change scenarios or execution length. The distinction is that the observed failures depend on the seed environments and rollout policy, whereas environment evolution explicitly edits the three requirement-level factors without requiring target-policy rollouts. These factors provide controllable evolution directions; the formulation does not assume that their effects are statistically independent.

E Additional Evolution and Training Diagnostics

E.1 Environment Mutation Across Generations

Figure 6 separates generation-level mutation rates into instruction, environment, and verification system components. Instruction mutation remains high under `high` and `max`, with both fluctuating between 95% and 100% while trending downward, whereas environment and verification system changes remain more selective.

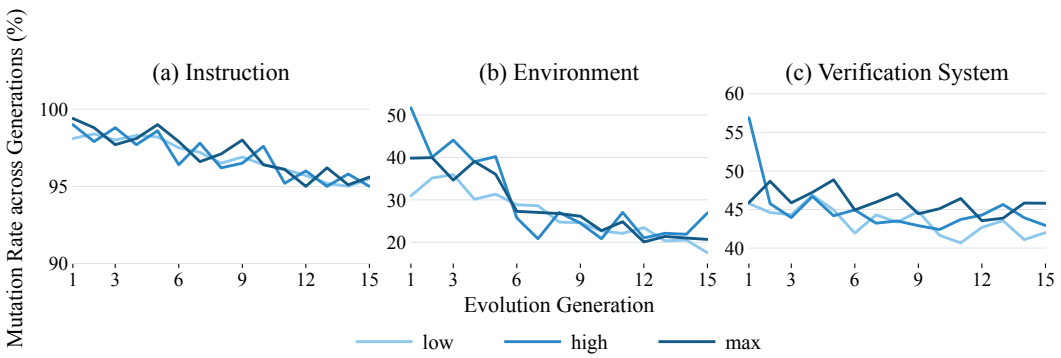


Figure 6: Instruction, environment, and verification system mutation rates across generations G1–G15 under `low`, `high`, and `max` evolution effort.

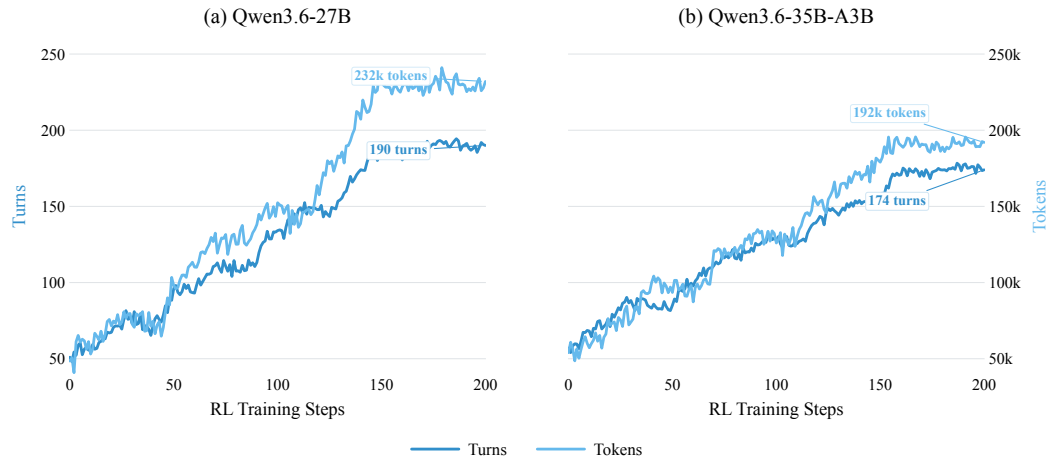


Figure 7: Trajectory length over 200 RL training steps for Qwen3.6-27B (left) and Qwen3.6-35B-A3B (right); the left and right y-axes report average turns and tokens per trajectory, respectively, with annotations at step 200. Tokens per turn increase from approximately 951 to 1,221 for Qwen3.6-27B and from 947 to 1,103 for Qwen3.6-35B-A3B.

E.2 Turns and Tokens During RL Training

During training, the scheduler progressively admits harder environments, which tend to require longer interaction horizons. As shown in Figure 7, this progression is accompanied by increases in both turns and tokens per trajectory. Tokens per turn also increase, suggesting that the policy allocates greater thinking effort to each interaction.