SEAL: SYNERGISTIC CO-EVOLUTION OF AGENTS AND LEARNING ENVIRONMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Model (LLM) agents increasingly learn through interaction, yet their training process often evolves unevenly: the policy changes with experience while the learning environment continues to provide largely fixed forms of support. As training progresses, this creates a growing mismatch between what the agent currently struggles with and what its learning process emphasizes. We call this phenomenon *Agent–Learning Environment Misalignment* (ALEM). We introduce SEAL, a framework for *synergistic co-evolution* of agents and learning environments. SEAL turns verifier-grounded execution failures into a shared diagnostic signal that coordinates two complementary forms of adaptation: it reshapes the support available in subsequent interactions and adjusts how the policy learns from collected trajectories. In this way, the learning environment follows the agent's evolving failure profile while both sides remain grounded in the same executable feedback. Across three backbones on BFCL V3, SEAL consistently improves over matched Vanilla RL, reaching 40.25% versus 30.75% with Qwen2.5-7B. A matched factorial ablation further yields a +5.00-point interaction estimate for the coupled configuration. The same trained checkpoints also show positive transfer to the evaluated BFCL V4 and τ^2-bench domains. These results suggest that coordinating how an agent learns with the environment it learns from is a promising direction for self-evolving tool-use agents.

1 INTRODUCTION

Ask an LLM agent for the cost of an economy flight from Rivermist to Stonebrook. It guesses the airport codes, calls `get_flight_cost` with `travel_from="RVM"` and `travel_to="STB"`, and the environment answers with a generic execution error, *No available route for the given airports*; the agent concludes that no route exists and gives up (Figure 4). Under standard reinforcement learning (RL), that is the whole lesson: the rollout receives reward zero, and the next rollout is collected under exactly the same conditions. Nothing records whether the agent skipped the airport-code lookup, passed an invalid argument, or failed to recover, so neither the policy update nor the environment can act on the difference. This loop is now a dominant way LLM agents improve. Beyond their capabilities in reasoning, planning, and tool use (Yao et al., 2023; Schick et al., 2023; Shinn et al., 2023; Chen et al., 2026; Yu et al., 2026b), agents are refined through RL, tool-use post-training, agentic data generation, and multimodal data evolution (Qian et al., 2025; Wei et al., 2026; Hu et al., 2025; Gao et al., 2025), and more generally through *self-evolution through interaction*: collecting rollouts, receiving feedback, and improving via reflection, RL, self-generated supervision, or skill memory (Wang et al., 2024; Zhai et al., 2025; Guo et al., 2025b; Sun et al., 2026; Wang et al., 2026). In realistic tool-use settings with multi-turn interfaces, strict execution constraints, and partial failures (Li et al., 2023; Xie et al., 2024; Liu et al., 2025a; Yin & Du, 2026; Peng et al., 2026; Yang et al., 2026b), improvement depends not only on how the policy is optimized but also on what learning signal the training-time environment exposes while the policy changes.

Existing self-evolving agents emphasize different parts of this loop. *Model-centric* methods improve the policy through rollout replay, self-reflection, reward optimization, or post-training (Huang et al., 2026; Yuan et al., 2025; Wang et al., 2025; Xiang et al., 2026), but against a fixed environment, so the learning signal is a function of the policy's own rollouts; with sparse rewards and long horizons, this self-referential signal produces exploration bias, unstable recovery, and inefficient credit assignment (Shridhar et al., 2021). *Environment-centric* methods instead adapt curricula, task distri-

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.

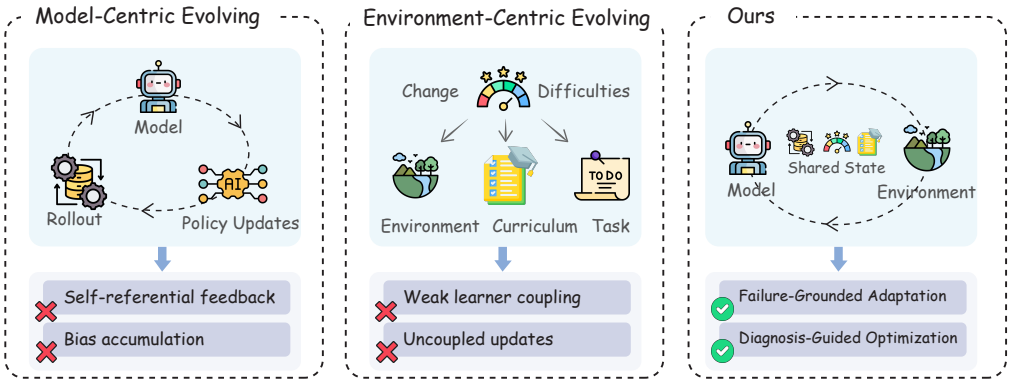


Figure 1: **Three ways to self-evolve a tool-use agent.** Left and middle illustrate model-centric and environment-centric designs with uncoupled updates. SEAL (right) explicitly couples complementary agent-side and learning-environment-side adaptations through a shared verifier-grounded diagnosis, with tasks, tools, rewards, and the verifier unchanged.

butions, synthetic instructions, or interaction experiences (Bengio et al., 2009; Lu et al., 2026; Yang et al., 2026a; Hao et al., 2026), including methods that use rollout or failure feedback. Our focus is to couple training-interface adaptation and policy reweighting through a shared executable diagnosis. We study a training-time gap we call **Agent–Learning Environment Misalignment (ALEM)**: *as the agent's capability frontier shifts during training, the learning environment that supplies interaction support and supervision does not track newly revealed failures, so the signal stays static, weakly targeted, or uninformative*. Throughout, "environment" means the training-time learning substrate (task exposure, observation interfaces, action constraints, recovery feedback), never the evaluation benchmark, tool semantics, or verifier. Closing the loop raises two challenges, both visible in the flight example. ❶ *Uninformative feedback*: the environment tells the agent *that* a call failed, not *why*, so the specific error cannot be repaired. ❷ *Undifferentiated credit*: a terminal scalar reward gives every failed rollout the same zero, so optimization pressure cannot be aimed at failures that are attributable and recoverable (a wrong argument type) rather than at failures that only surface in the final response check.

We propose **SEAL**, a closed-loop co-evolution framework for multi-turn tool-use agents (Figure 1). Here, *synergistic* denotes the coupled design: shared diagnosis coordinates the current policy update and subsequent interaction support; Section 4.4 examines their empirical complementarity. SEAL diagnoses every rollout with executable evidence (parser, schema, execution, state, and verifier checks) into turn-level labels such as a missing call or a state mismatch. For ❶, the training-time interface evolves to expose schema affordances, constraints, and recovery-oriented feedback selected by the agent's recurring failures. For ❷, policy optimization reweights each trajectory's GRPO advantage by how attributable and recoverable its failures are. The environment thus adapts to the current policy's failures, not a difficulty schedule. Crucially, SEAL never touches the benchmark: tasks, tools, rewards, and the verifier stay fixed; the interface only re-exposes existing information, never reference calls, hidden parameters, or answers; and every training-only addition is removed at evaluation, so SEAL and its baselines are tested under identical conditions. This conservatism lets a gain be attributed to a better policy rather than a friendlier test. Because diagnosis needs executable evidence, we study interactive tool use, where such evidence is structured and auditable. Our contributions are threefold:

- We study **ALEM**, the training-time mismatch between an evolving agent's execution failures and its learning environment, and trace it to two challenges: uninformative feedback (❶) and undifferentiated credit (❷).
- We introduce **SEAL**, where a shared verifier-grounded diagnosis coordinates learning-interface evolution and diagnosis-guided advantage reweighting, while keeping tasks, tools, rewards, and the verifier fixed.
- We show that SEAL improves over matched Vanilla RL across three backbones (+9.50 points at 7B), exceeds Environment Tuning in reported point estimates, and transfers positively to the evaluated held-out domains; a matched factorial ablation yields a +5.00-point interaction estimate.

2 Related Work

One-sided self-evolution. Model-side methods refine the agent through skill learning, self-consolidation, prompt adaptation, memory, or RL from interaction feedback (Xia et al., 2026; Yu et al., 2026a; Agrawal et al., 2026; Zweiger et al., 2025): experience is internalized into the policy, prompt, memory, or skills while the training-time environment stays fixed, so the feedback is only as informative as that environment makes it. Environment-side methods instead reshape what the learner sees through curricula, synthetic instructions, task evolution, or tool and skill construction (Matiisen et al., 2020; Portelas et al., 2020; Wang et al., 2023; Xu et al., 2024; Cai et al., 2024; Qian et al., 2023; Yuan et al., 2024), acting on task diversity, difficulty, or coverage, driven by task-level success rather than by a diagnosis of why the current policy fails. In this work, we keep the model-side update and make environment adaptation *failure-conditioned*: verifier-grounded diagnoses decide which cues and recovery feedback the current policy sees.

Environments and closest methods. Interactive benchmarks for tool use, web navigation, operating-system control, embodied simulation, and software engineering expose the multi-turn dependencies, sparse rewards, and recovery dynamics SEAL targets (Patil et al., 2025; Li et al., 2023; Zhou et al., 2024; Liu et al., 2024; Jimenez et al., 2024); recent work co-evolves agents with generated or edited environments (Xi et al., 2025; Zhang et al., 2025; Guo et al., 2025a), whereas SEAL keeps tasks, tool semantics, and the verifier fixed and co-evolves only the training-time interface. ALIGN (Liu et al., 2025a) augments interfaces for agent–environment alignment; SEAL instead uses verifier-grounded diagnoses to adjust training-time support and policy updates, addressing ALEM. Closest in mechanism, Dynamic Cheatsheet (Suzgun et al., 2026) evolves an inference-time memory without updating parameters, whereas SEAL's guidance exists only at training time; Fission-GRPO (Zhang et al., 2026) turns failures into corrective instances with a learned error simulator and extra rollouts, whereas SEAL reweights existing rollouts with deterministic diagnosis; and Environment Tuning (Lu et al., 2026), our closest method-level comparison (Table 2), adapts task exposure, feedback, and progress supervision, whereas SEAL keeps the binary verifier reward and changes only the interface and optimization weights.

3 Methodology

Figure 2 gives an overview of SEAL. Standard RL gives a tool-use agent one binary reward per episode; SEAL keeps that reward but, during training, also **diagnoses every rollout turn by turn from executable evidence**. These diagnoses are used twice: they decide *which feedback* the training-time interface adds for the agent, such as recovery hints, and *how much* each trajectory counts in the policy update. We take the view of a trainer who controls training but not evaluation, and separate the *tool backend*, which defines what a call does, from the *learning interface*, which defines what the agent observes. SEAL changes **only the learning interface**: tools, task labels, rewards, and the verifier never change, so the benchmark still measures the same thing.

3.1 Problem Formulation

We model interactive tool use as a partially observable decision process with hidden states, horizon H, and a tool set $\mathcal{F}$: actions include natural-language responses and executable tool calls, and observations include dialogue context, tool outputs, and execution errors. Given an instruction $q \sim \mathcal{D}$, a policy π_θ interacting with an executable environment $\mathcal{E}$ induces a rollout

$$\tau = \{(u_i, a_i, o_i)\}_{i=1}^{T} \sim P_{\mathcal{E}}(\cdot \mid q, \pi_\theta), \qquad T \le H, \tag{1}$$

where u_i is the dialogue context, a_i the model action, and o_i the environment observation. The executable verifier $\mathcal{V}$ returns a terminal binary reward $r(\tau) = \mathcal{V}(\tau, q, \mathcal{E}) \in \{0, 1\}$, and standard RL maximizes $J(\pi_\theta) = \mathbb{E}_{q\sim\mathcal{D},\, \tau\sim P_{\mathcal{E}}(\cdot|q,\pi_\theta)}[r(\tau)]$. This scalar says whether a trajectory succeeded, not why it failed. SEAL therefore augments the training-time feedback to $(r(\tau), Z(\tau))$, where $Z(\tau)$ is a structured failure diagnosis extracted from the executable interaction trace; the reward and evaluation protocol are unchanged.

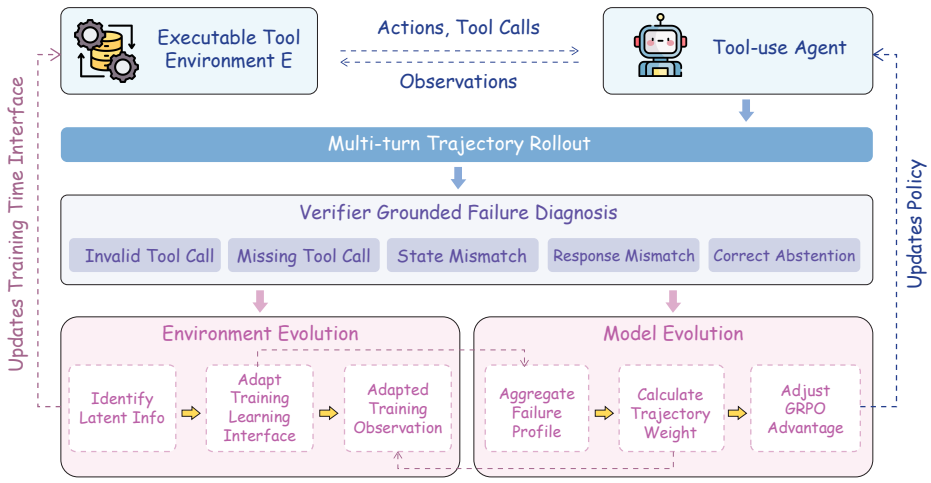


Figure 2: **The SEAL training loop.** The agent interacts with the executable tool environment to produce multi-turn trajectories (top), which executable evidence and the verifier diagnose into turn-level failure labels (middle). The same labels drive environment evolution (bottom left: identify the missing information, adapt the training-time interface, expose an adapted observation in the next round) and model evolution (bottom right: aggregate the failure profile, compute trajectory weights, rescale the GRPO advantages of the current update), while tool semantics, rewards, and evaluation never change. The diagnosis block shows behavioral categories; the operational labels are in Table 1.

Table 1: **Operational diagnosis labels.** Ψ evaluates the conditions in order and assigns the first that holds; acceptance is defined by the original verifier. ρ_0 is the base utility of Section 3.4: labels with concrete executable evidence and a clear repair direction weigh more than downstream verifier-level mismatches.

Order	Label	Executable condition	ρ_0
1a	`correct_abstention`	No tool call is required, and the model makes none.	1.8
1b	`spurious_tool_call`	No tool call is required, but the model attempts one.	2.0
2	`empty_turn_model_response`	A tool call is required, but none is produced.	1.5
3	`instance_mismatch`	Model-side and reference-side executor object sets differ.	0.6
4	`state_mismatch`	The official state checker rejects the resulting state.	1.0
5	`response_mismatch`	State passes, but the execution-response checker rejects.	0.9
6	`pass`	Both state and execution-response checks pass.	1.0

3.2 Verifier-Grounded Failure Diagnosis

For each rollout, SEAL produces turn-level labels $Z(\tau) = \{z_1, \ldots, z_T\}$, $z_i \in \mathcal{Z}$, where z_i is the dominant outcome or failure mode at turn i:

$$z_i = \Psi(a_i, \xi_i, s_{i-1}, s_i, \mathcal{F}), \tag{2}$$

with ξ_i the executable evidence at turn i and s_{i-1}, s_i the pre- and post-action states, when available.

Rules, not a judge. Ψ is a deterministic classifier over executable traces (parser checks, schema validation, execution errors, state transitions, verifier comparisons), applied in a fixed priority order that puts directly executable failures before downstream verifier-level ones (Appendix E.2). We use rules rather than an LLM judge because executable checks are trace-grounded and reproducible, whereas a learned judge can reintroduce the self-referential feedback of model-centric evolution.

Seven labels. Table 1 lists the labels, their executable conditions (checked top to bottom), and their base utilities ρ_0. `response_mismatch` is the BFCL execution-response check, not the wording of a final answer, and bookkeeping outcomes (`checker_error`, `gt_error`) are excluded from weighting and recovery selection.

Reliability. Deterministic rules can still mislabel, so we audited them: in a blinded audit of 60 label-stratified diagnoses (Appendix H.2), the annotator agreed with Ψ on the exact label in 57 cases (95.0%) and on failure versus non-failure in 59, with every disagreement on state versus execution-response matching. Diagnosis never touches the reward: a failed trajectory still receives $r(\tau) = 0$.

3.3 LEARNING-INTERFACE EVOLUTION

SEAL evolves only the *training-time learning interface*, never the verifier, tool signatures, tool outputs, or task labels. At turn i it augments the original observation o_i as

$$\tilde{o}_i = \Omega_t(o_i, \mathcal{F}, H_t, C_{t-1}) = o_i \oplus \phi_{\text{schema}}(\mathcal{F}) \oplus \phi_{\text{err}}(o_i, H_t) \oplus \phi_{\text{cap}}(C_{t-1}), \tag{3}$$

where H_t comes from previous rollouts and C_{t-1} is the latest completed failure profile (empty at $t = 0$); this round's C_t affects Ω_{t+1} only after sampling. Each component is a deterministic transformation or fixed template, not an LLM:

- **Schema cues** (ϕ_{schema}) make schema constraints explicit (required arguments, enums, types, call formats), because most invalid calls violate a constraint the schema already states.
- **Recovery feedback** (ϕ_{err}) turns a generic execution error into a hint on how to recover, without revealing the answer, because a bare error is the uninformative feedback of Section 1.
- **Capability cues** (ϕ_{cap}) add a class-level instruction for each recurring failure: cautious calls for `spurious_tool_call`, an active response for `empty_turn_model_response`, and state-aware recovery for `state_mismatch` (templates in Appendix F).

Safeguards. The cues describe how to repair an error class, never the reference calls, hidden parameters, or final answer, and the whole interface is removed at evaluation. Proposition 3 (Appendix D) shows that, for unrestricted policies and append-only observations, accepted trajectories and optimal-policy availability are preserved. For a finite set $\mathcal{L}$ of complete template configurations, the additional single-round signal carries at most $\log_2 |\mathcal{L}|$ bits about the reference conditioned on raw observations and schema; this does not guarantee absence of leakage.

3.4 DIAGNOSIS-GUIDED ADVANTAGE REWEIGHTING

Under a binary reward, GRPO (Shao et al., 2024) gives every failed trajectory in a group the same advantage, $-\sqrt{k/(G-k)}$ for k successes among G rollouts (Lemma 1, Appendix D), so an invalid argument and a wrong final answer are pushed down equally. SEAL instead weights each trajectory by what its diagnoses reveal, in three steps.

Diagnostic profile. For trajectory τ_j with valid, non-bookkeeping turn labels $\mathcal{I}_j$, the profile $p_j(z) = \frac{1}{|\mathcal{I}_j|} \sum_{i \in \mathcal{I}_j} \mathbf{1}[z_i = z]$ is the share of its turns carrying label $z \in \mathcal{Z}$.

Label utilities. A utility $\rho_t(z) > 0$ scores how actionable and attributable label z is. The fixed base utilities ρ_0 (Table 1) favor failures with concrete executable evidence and a clear repair, such as `spurious_tool_call`, over ambiguous ones such as `response_mismatch`; `pass` stays at 1.0. For a failure label z, let $n_t(z)$ count its occurrences in the statistics window, $p_t(z)$ be its share of non-bookkeeping occurrences, $\bar{r}_t(z)$ the mean verifier reward of the trajectories containing it, and $c_t(z) = \min(1, n_t(z)/32)$ a support factor. The realized utilities then adapt as

$$\rho^*_t(z) = \text{clip}_{[0.5,2.0]} \left[\rho_0(z)\{1 + p_t(z)(1 - \bar{r}_t(z))c_t(z)\}\right], \tag{4}$$

$$\rho_t(z) = 0.75\, \rho_{t-1}(z) + 0.25\, \rho^*_t(z), \tag{5}$$

so a label gains weight while it is frequent and rarely co-occurs with success, while low-support labels exert smaller utility updates. By Proposition 1, utilities stay in $[0.5, 2.0]$ and move by at most 0.375 per round; when a failure type disappears, the EMA gradually relaxes its utility toward ρ_0. All constants are fixed by the comparison protocol.

Weighted advantage. Each trajectory's weight averages the utilities over its profile,

$$w_j = \text{clip}\Big(\sum_{z \in \mathcal{Z}} p_j(z)\, \rho_t(z),\ w_{\min},\ w_{\max}\Big), \qquad [w_{\min}, w_{\max}] = [0.5, 2.0], \tag{6}$$

with $w_j = 1$ when no valid label exists (the clip never binds, Corollary 1). SEAL scales the group-normalized GRPO advantage to $\widetilde{A}_j = w_j A_j$ on all trainable response tokens, for positive and negative advantages alike, without scaling the reward, adding token-level weights, or renormalizing. Because $w_j > 0$, weighting preserves each group-relative advantage's sign and no sampled trajectory is discarded. The weighted reward-driven score-function term admits a tilted-distribution representation with a bounded normalization factor and density ratio to π_θ in $[1/4, 4]$ (Proposition 2); this does not equate the full clipped, KL-regularized GRPO update to an unweighted one.

3.5 TRAINING LOOP

Each round is one optimizer iteration over 32 prompts with eight rollouts each (Algorithm 1, Appendix C) and runs four steps: (1) **collect** trajectories $\{\tau_j\}_{j=1}^{N}$ by running π_{θ_t} under interface Ω_t; (2) **diagnose** each τ_j into its verifier reward $r(\tau_j)$ and labels $Z(\tau_j)$, and aggregate the failure profile $C_t = \mathrm{Agg}(\{Z(\tau_j)\})$; (3) **update the policy** on this batch, $\theta_{t+1} = \mathrm{GRPO}(\theta_t; \{\tau_j, r(\tau_j), w_j A_j\})$; and (4) **evolve the interface** for the next round, $\Omega_{t+1} = \mathrm{Evolve}(\Omega_t, C_t)$. Diagnoses never alter trajectories already sampled, and tools, task labels, rewards, and the verifier stay fixed throughout. The two sides also cover each other's blind spot: reweighting redistributes credit only within groups that contain both successes and failures, whereas the interface changes the rollouts of prompts the policy never solves (Remark 1, Appendix D).

4 EXPERIMENTS

We test, through four questions, our central claim: a tool-use agent learns more from the same training data when it co-evolves with its learning environment. **RQ1**: Does SEAL improve in-distribution multi-turn tool use over RL with the same training examples and rollout budget (Section 4.2)? **RQ2**: Do the gains transfer to held-out tool domains (Section 4.3)? **RQ3**: Do the two diagnosis-driven adaptations provide complementary gains when coupled (Section 4.4)? **RQ4**: How do the gains emerge during training (Section 4.5)?

4.1 EXPERIMENTAL SETUP

Benchmarks. In-distribution: the BFCL V3 multi-turn subset (Patil et al., 2025), 800 examples in four categories (*Base*, *Missing Functions*, *Missing Parameters*, *Long Context*), split at the scenario-ID level (seed 42) into 400 training and 400 task-disjoint evaluation examples, 100 per category (Appendix A). Out-of-distribution: the BFCL V4 Web Search and Memory tracks and the Retail, Airline, and Telecom domains of τ^2-bench (Barres et al., 2025), which differ from BFCL V3 in tool domains, schemas, and interaction patterns. All evaluations use the original benchmark environments and verifiers.

Backbones. We train Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct (Yang et al., 2024), general instruction-tuned agents, and ToolACE-2-Llama-3.1-8B (Liu et al., 2025b), a stronger tool-specialized initialization that tests whether the method still helps a model that already calls tools well. The proprietary and open-source reference systems in Tables 2 and 3 are described, with references, in Appendix B.

Conditions. All binary-reward conditions share the backbone, the 400-example split, eight rollouts per prompt, GRPO and optimizer settings, and the binary verifier reward, and are evaluated with every training-only addition removed. VANILLA RL is GRPO with the original interface (the matched training baseline). SEAL adds training-interface enrichment and diagnosis-guided advantage reweighting; the component settings are described in Appendix H.1. Environment Tuning (Lu et al., 2026), the closest published method (curriculum, hand-written error hints, per-turn progress reward), is trained on the same split with its released recipe; it is a method-level rather than reward-matched comparison. Optimizer settings are listed in Table 5. The training comparisons report one run per condition. The main Qwen2.5-7B SEAL checkpoint uses the complete online configuration: fixed schema affordances, diagnosis-conditioned recovery selection, and diagnosis-aware advantage reweighting. Diagnoses reweight the current policy update and select recovery guidance for subsequent training rounds; all adaptive guidance is removed at evaluation (Appendix E.5).

4.2 IN-DISTRIBUTION RESULTS ON BFCL V3 (RQ1)

SEAL beats RL on every reported backbone and category, with higher point estimates than the closest environment-side method. Table 2 supports this from five angles. *Consistency*: under the same rollouts, optimizer, and reward, SEAL exceeds VANILLA RL in all 12 backbone-by-category cells, so the gain is not carried by a favorable subset. *Magnitude*: its total improvement over the untrained checkpoint is 1.6 to 2.4 times that of RL alone, reaching +26.25 points on Qwen2.5-7B, which suggests that a sparse terminal reward leaves useful learning signal unused. *Initialization*:

Table 2: **Main results on the BFCL V3 multi-turn benchmark (accuracy, %).** Our training comparisons use 400 examples and report one run per condition. The Qwen2.5-7B SEAL row uses the complete online configuration (Appendix E.5). Parentheses give gains over the untrained backbone. Reference systems use different scales, data, and recipes (Appendix B).

Model	BFCL V3 Multi-Turn				
	Average	Base	Miss Func	Miss Param	Long Context
Closed-source					
Gemini-3-Pro-Preview	60.75	64.50	60.00	54.50	64.00
Claude-Sonnet-4.5	61.38	69.00	65.00	52.50	59.00
GPT-4o-2024-11-20	42.50	55.50	34.50	29.00	51.00
Gemini-2.5-Flash	36.25	41.50	36.00	32.00	35.50
Open-source					
GLM-4.6 355B	68.00	74.50	68.00	63.00	66.50
Qwen3-32B	47.88	56.00	52.50	40.00	43.00
Qwen2.5-14B-Instruct	25.25	33.00	26.00	22.00	20.00
xLAM-2-3b-fc-r	58.38	71.50	59.00	57.50	45.50
Qwen2.5-3B-Instruct	5.75	11.00	6.00	3.00	3.00
+ VANILLA RL	9.25 (+3.50)	16.00 (+5.00)	9.00 (+3.00)	6.00 (+3.00)	6.00 (+3.00)
+ Environment Tuning	11.50 (+5.75)	16.00 (+5.00)	10.00 (+4.00)	8.00 (+5.00)	12.00 (+9.00)
+ SEAL	14.00 (+8.25)	19.00 (+8.00)	15.00 (+9.00)	12.00 (+9.00)	10.00 (+7.00)
Qwen2.5-7B-Instruct	14.00	22.00	14.00	10.00	10.00
+ VANILLA RL	30.75 (+16.75)	46.00 (+24.00)	27.00 (+13.00)	27.00 (+17.00)	23.00 (+13.00)
+ Environment Tuning	37.00 (+23.00)	52.00 (+30.00)	34.00 (+20.00)	35.00 (+25.00)	27.00 (+17.00)
+ SEAL	40.25 (+26.25)	58.00 (+36.00)	36.00 (+22.00)	34.00 (+24.00)	33.00 (+23.00)
ToolACE-2-Llama-3.1-8B	32.00	45.00	26.00	35.00	22.00
+ VANILLA RL	38.50 (+6.50)	52.00 (+7.00)	30.00 (+4.00)	40.00 (+5.00)	32.00 (+10.00)
+ Environment Tuning	43.25 (+11.25)	54.00 (+9.00)	41.00 (+15.00)	36.00 (+1.00)	42.00 (+20.00)
+ SEAL	46.75 (+14.75)	58.00 (+13.00)	46.00 (+20.00)	44.00 (+9.00)	39.00 (+17.00)

the tool-specialized ToolACE-2 also gains substantially over its RL counterpart, suggesting that diagnosis-guided training complements tool-specific pre-training. *Failure type*: averaged across backbones relative to Vanilla RL, *Missing Functions* improves most, *Long Context* ties *Base*, and *Missing Parameters* improves least, plausibly because recognizing an underspecified argument depends on the user's instruction rather than the tool schema. *Method-level baseline*: SEAL has higher point estimates than Environment Tuning on every backbone (by 2.50 to 3.50 points) and in 9 of 12 cells while needing none of its curriculum, hand-written per-domain hints, or dense progress reward; SEAL keeps the binary reward and derives its cues from diagnosed failure classes. Against the reference rows, 400 examples take Qwen2.5-7B past Gemini-2.5-Flash and its own 14B sibling, and ToolACE-2 with SEAL past GPT-4o to within about one point of the four-times larger Qwen3-32B, though the strongest systems remain well ahead. In short, diagnosis-driven co-evolution recovers learning signal that RL alone leaves on the table, across scale, initialization, and failure type; we next ask whether the gains are specific to BFCL V3.

4.3 OUT-OF-DISTRIBUTION TRANSFER (RQ2)

SEAL shows positive transfer in the evaluated held-out tool domains. Table 3 evaluates the BFCL V3-trained Qwen2.5 models on BFCL V4 and τ^2-bench, whose tools never appear in training, and four angles show that the gains carry over. *Comparison*: SEAL's gains on the held-out averages are 2.2 to 4.3 times those of VANILLA RL, which moves them by about one point. *Breadth*: SEAL improves all ten backbone-by-track cells over the untrained model and matches or beats VANILLA RL in each, even on Web Search, whose search tools have no counterpart in BFCL V3. *Magnitude*: the largest gain comes in a domain absent from training, τ^2-bench Retail, where Qwen2.5-7B rises from 16.67% to 26.32%, more than three times the gain of RL alone. *Mechanism*: SEAL's cues and credit target failure classes (for instance, changing state without checking it first) rather than BFCL tasks, and the cues are removed at evaluation; these scores provide preliminary evidence that BFCL V3 gains transfer to the evaluated held-out domains. Absolute scores are still low, and Web Search

Table 3: **Out-of-distribution transfer to BFCL V4 and τ^2-bench (accuracy, %).** SEAL and Vanilla RL reuse the Table 2 checkpoints without OOD-specific training or selection; each was trained on 400 BFCL V3 examples and reports one run. Parentheses give gains over the untrained backbone. Reference rows are not matched training controls (Appendix B).

Model	BFCL V4			τ^2-bench			
	Avg.	Web Search	Memory	Avg.	Retail	Airline	Telecom
BitAgent-8B	7.41	4.50	10.32	10.00	7.50	15.00	7.50
ToolACE-2-Llama-3.1-8B	15.90	9.00	22.80	10.83	10.00	15.00	7.50
watt-tool-8B	8.67	4.00	13.33	13.33	15.00	20.00	5.00
Qwen3-30B-A3B-Instruct	20.07	22.50	17.63	32.03	50.88	32.50	12.72
Qwen2.5-3B-Instruct	4.69	0.00	9.38	9.64	5.26	14.00	9.65
+ VANILLA RL	5.88 (+1.19)	0.50	11.26	10.60 (+0.96)	6.14	16.00	9.65
+ SEAL	8.63 (+3.94)	2.00	15.26	11.76 (+2.12)	8.77	16.00	10.52
Qwen2.5-7B-Instruct	9.17	6.50	11.83	16.11	16.67	20.00	11.67
+ VANILLA RL	10.26 (+1.09)	7.50	13.02	17.19 (+1.08)	19.30	20.00	12.28
+ SEAL	12.71 (+3.54)	9.50	15.92	20.79 (+4.68)	26.32	22.00	14.04

Table 4: **Factorial ablation of interface adaptation and diagnosis-aware optimization.** Qwen2.5-7B, BFCL V3 multi-turn accuracy (%). Each condition reports one run; the interaction is a point estimate. Full SEAL uses Table 2's checkpoint. Δ gives differences from Full SEAL in percentage points.

Interface	Weighting	Avg.	Δ	Base	Miss Func	Miss Param	Long Context
Static	Uniform	34.75	-5.50	52.00	32.00	32.00	23.00
Dynamic	Uniform	34.50	-5.75	49.00	34.00	31.00	24.00
Static	Diagnosis-aware	35.50	-4.75	50.00	35.00	32.00	25.00
Dynamic	Diagnosis-aware	**40.25**	—	58.00	36.00	34.00	33.00

stays near zero at 3B, so cross-benchmark generalization remains hard. In short, the observed gains extend to the evaluated held-out benchmarks; we next ask which parts of the system are responsible.

4.4 ABLATION STUDY (RQ3)

In this run, the two sides work best together. Table 4 crosses interface selection and advantage weighting on Qwen2.5-7B. *Hierarchy*: replacing diagnosis-aware weighting costs 5.75 points and replacing dynamic interface selection costs 4.75, yet every variant still beats VANILLA RL. *Complementarity*: relative to Static+Uniform, diagnosis-aware weighting alone changes accuracy by +0.75 points and dynamic interface selection alone by -0.25; together they add 5.50, yielding a positive interaction point estimate of +5.00 points. This is consistent with Remark 1: the interface can turn unsolved prompts into the mixed success-failure groups on which reweighting acts. *Adaptivity*: fixing the interface and using uniform weighting gives up 5.50 points of the full configuration's accuracy. *Consistency*: full SEAL is best in every category, and no variant falls below VANILLA RL in any. In this single-run comparison, the pattern is consistent with complementary adaptations coordinated through a shared execution diagnosis.

4.5 TRAINING DYNAMICS (RQ4)

SEAL learns faster and keeps improving where plain RL plateaus. *Timing*: in Figure 3, both runs rise together for about 30 steps; then VANILLA RL peaks near step 60 and slips back, while SEAL keeps climbing, matches RL's peak by mid-training, and ends about ten points ahead. *Theory*: by Lemma 1, groups with identical binary verifier rewards have zero group-relative reward advantages, so diagnosis weighting gives them no reward-driven GRPO policy-gradient signal; KL and other auxiliary terms may still contribute gradients, while SEAL's interface can change later rollouts of those prompts. *Diagnosis*: in a separate implementation-variant run (Appendix H.3), model-error label occurrences fall by over a third, led by response and state mismatches, while spurious-tool-call label counts stay approximately flat. *Cost*: in a separate timing comparison, the online-interface

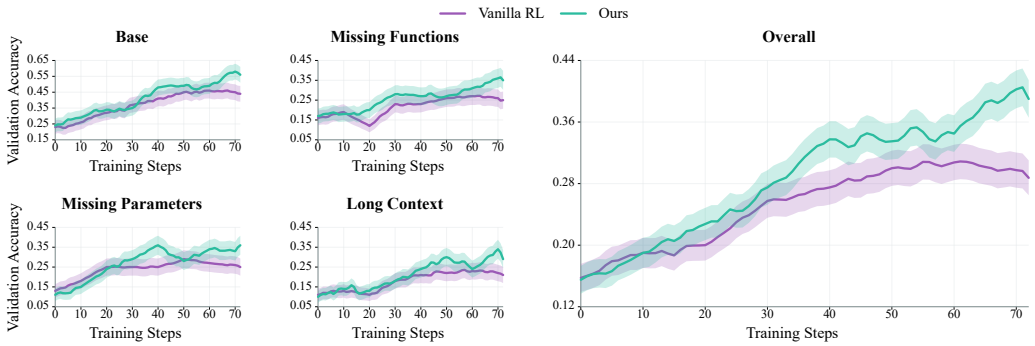


Figure 3: **Training dynamics of VANILLA RL and SEAL on the BFCL V3 validation subsets.** Validation accuracy across training steps per category and overall. Shading is the recorded binomial approximation, not training-seed variation or an interval for paired differences; the overall band does not account for dependence among variants of the same scenario.

Base model (reward 0)

User: economy flight cost, Rivermist to Stonebrook, 2026-10-06.

Assistant → `get_flight_cost(travel_from="RVM", travel_to="STB", travel_date=2026-10-06)`

Tool: `Error: No available route for the given airports.`

Assistant: "There is no direct flight route available."

User: it is 15400 GBP; my access token is `abc123token`.

Assistant → `set_budget_limit(15400.0)` ⇒ **Tool:** `{"budget_limit": 15400.0}` (GBP stored as USD)

SEAL-trained model (reward 1)

User: economy flight cost, Rivermist to Stonebrook, 2026-10-06.

Assistant → `get_flight_cost(travel_from="RVM", travel_to="STB", travel_date=2026-10-06)`

Tool: `Error: No available route for the given airports.`

Assistant → `get_nearest_airport_by_city("Rivermist")`, `("Stonebrook")` ⇒ **Tool:** `"RMS", "SBK"`

Assistant → `get_flight_cost("RMS", "SBK", 2026-10-06, economy)` ⇒ **Tool:** `[220.0]`

User: it is 15400 GBP; my access token is `abc123token`.

Assistant → `compute_exchange_rate(GBP, USD, 15400)` ⇒ **Tool:** `22000.0`

Assistant → `set_budget_limit(22000.0)` ⇒ **Tool:** `{"budget_limit": 22000.0}`

Figure 4: **Before and after on BFCL `multi_turn_miss_param_150` (Qwen2.5-7B).** Same task, tools, and verifier; only the policy differs. **Assistant** → lines are tool calls and **Tool** entries their abbreviated responses (one user turn omitted); the SEAL trajectory matches the expected outcome (Appendix G.2).

configuration has 7.1% higher median step time than Vanilla RL (Appendix H.4). In short, the same sparse reward keeps paying off under SEAL well after RL stalls.

4.6 CASE STUDY

The SEAL-trained agent recovers where the base model gives up. Figure 4 follows the BFCL task of Section 1 to the end. The base model fails twice: it gives up at the generic route error, leaving a wrong execution response, and later stores a GBP amount as a USD budget, leaving a wrong final state. The SEAL-trained model repairs both and turns reward 0 into 1: it looks up the airport codes and retries, then converts the currency before setting the budget. This behavior illustrates the class-level recovery cue ("resolve the stated prerequisite before retrying", Appendix F), now followed without the cue at evaluation. Four further cases are in Appendix G.

5 CONCLUSION

This paper argues that a self-improving tool-use agent needs a learning environment that improves with it. We address Agent–Learning Environment Misalignment (ALEM) with SEAL: a shared verifier-grounded diagnosis coordinates adaptive training support and diagnosis-aware policy updates, leaving the benchmark untouched. Across three backbones, this design improves over matched Vanilla RL; the factorial ablation yields a positive interaction point estimate, and the same checkpoints transfer positively to the evaluated held-out domains. In short, a diagnosis is a signal for the teacher as much as for the learner.

AI USE STATEMENT

Generative AI assisted experimental planning and refinement, implementation and debugging, analysis and interpretation of rollout records, literature search, and manuscript drafting and editing during development and revision. An AI assistant also produced the first-pass annotations for the 60-case blinded diagnosis audit in Appendix H.2, using dialogue, calls, initial state, and execution evidence while blinded to SEAL's label. We report this as AI-assisted agreement, not independent human annotation. Checks performed for this revision include recomputation of diagnostic summaries from stored records, source/configuration consistency checks, and LaTeX compilation. The training-time diagnoser itself remains deterministic and does not invoke an LLM judge. The authors are responsible for the paper's claims, results, code, and AI-assisted content.

ETHICS STATEMENT

This work studies tool-use agents in public benchmark environments (BFCL V3, BFCL V4, and τ^2-bench), rather than deployment with real customers. More capable agents can be misused, and improved benchmark performance does not establish safe deployment. Training guidance excludes explicit reference calls and hidden answer values, but the finite-template analysis is not a guarantee against information leakage. Benchmark and model use remains subject to the respective released licenses.

REPRODUCIBILITY STATEMENT

Appendices A–E describe the scenario-level split, algorithm, qualified theoretical properties, hyperparameters, and diagnostic rules. Appendix F supplies interface templates, and Appendix H describes the main comparisons and auxiliary analyses. The main Qwen2.5-7B configuration is specified in Appendix E.5; the auxiliary diagnosis audit's protocol is described in Appendix H.2. Code and split manifests are intended for release. The training comparisons report one run per condition and do not estimate variation across training seeds.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Anthropic. Claude Sonnet 4.5 system card. System card, 2025. https://www.anthropic.com/claude-sonnet-4-5-system-card.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 41–48, 2009.

BitAgent. BitAgent-8B model card. Hugging Face model card, 2025. https://huggingface.co/BitAgent/BitAgent-8B.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *The Twelfth International Conference on Learning Representations*, 2024.

Zhangquan Chen, Jiale Tao, Ruihuang Li, Yihao Hu, Ruitao Chen, Zhantao Yang, Xinlei Yu, Haodong Jing, Manyuan Zhang, Shuai Shao, Biao Wang, Qinglin Lu, and Ruqi Huang. OmniVideo-R1: Reinforcing audio-visual reasoning with query intention and modality attention. In *Forty-third International Conference on Machine Learning*, 2026.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Minghe Gao, Zhongqi Yue, Wenjie Yan, Yihao Hu, Wei Ji, Siliang Tang, Jun Xiao, Tat-Seng Chua, Yueting Zhuang, and Juncheng Li. Counterfactual evolution of multimodal datasets via visual programming. In *Advances in Neural Information Processing Systems*, volume 38, pp. 81947–81976, 2025.

Google. Gemini 3 Pro Preview model documentation. Google AI for Developers documentation, 2025. https://ai.google.dev/gemini-api/docs/models/gemini-3-pro-preview.

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. *arXiv preprint arXiv:2512.19682*, 2025a.

Yifu Guo, Jiaye Lin, Huacan Wang, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, and Mingguang Chen. SE-Agent: Self-evolution trajectory optimization in multi-step reasoning with LLM-based agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 116314–116341, 2025b.

Bingguang Hao, Zengzhuang Xu, Yuntao Wen, Xinyi Xu, Yang Liu, Tong Zhao, Maolin Wang, Long Chen, Dong Wang, Yicheng Chen, et al. From failure to mastery: Generating hard samples for tool-use agents. *arXiv preprint arXiv:2601.01498*, 2026.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jian-Guang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. AgentGen: Enhancing planning abilities for large language model based agent via environment and task generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, pp. 496–507, 2025.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-Zero: Self-evolving reasoning LLM from zero data. In *The Fourteenth International Conference on Learning Representations*, 2026.

Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world Github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116, 2023.

Kaiming Liu, Xuanyu Lei, Ziyue Wang, Peng Li, and Yang Liu. Agent-environment alignment via automated interface generation. *arXiv preprint arXiv:2505.21055*, 2025a.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. ToolACE: Winning the points of LLM function calling. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024.

Siyuan Lu, Zechuan Wang, Hongxuan Zhang, Qintong Wu, Leilei Gan, Chenyi Zhuang, Jinjie Gu, and Tao Lin. Don't just fine-tune the agent, tune the environment. In *The Fourteenth International Conference on Learning Representations*, 2026.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3732–3740, 2020.

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 *Proceedings of the 42nd International Conference on Machine Learning*, pp. 48371–48392. PMLR, 2025.

Hanli Peng, Yongsen Zheng, Ziyao Liu, and Kwok-Yan Lam. Tool execution hallucination in LLM-based agents: A unified taxonomy with detection, mitigation, and future directions. *TechRxiv*, 2026.

Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep RL: A short survey. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pp. 4819–4825, 2020.

Cheng Qian, Chi Han, Yi Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. CREATOR: Tool creation for disentangling abstract and concrete reasoning of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 6922–6939, 2023.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. ToolRL: Reward is all tool learning needs. In *Advances in Neural Information Processing Systems*, volume 38, pp. 105523–105553, 2025.

Salesforce AI Research. xLAM-2-3b-fc-r model card. Hugging Face model card, 2025. `https://huggingface.co/Salesforce/xLAM-2-3b-fc-r`.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. SEAgent: Self-evolving computer use agent with autonomous learning from experience. In *Forty-third International Conference on Machine Learning*, 2026.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

Pan Wang, Yihao Hu, Xiujin Liu, Jingchu Yang, Hang Wang, and Zhihao Wen. AtlasVA: Self-evolving visual skill memory for teacher-free VLM agents. *arXiv preprint arXiv:2605.17933*, 2026.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, 2023.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

watt-ai. watt-tool-8B model card. Hugging Face model card, 2024. `https://huggingface.co/watt-ai/watt-tool-8B`.

Tianxin Wei, Ting-Wei Li, Zhining Liu, Xuying Ning, Ze Yang, Jiaru Zou, Zhichen Zeng, Ruizhong Qiu, Xiao Lin, Dongqi Fu, et al. A survey of agentic reasoning for large language models: Towards recursively self-improving and collective agents. *arXiv preprint arXiv:2601.12538*, 2026.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. AgentGym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27914–27961, 2025.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Zhishang Xiang, Chengyi Yang, Zerui Chen, Zhimin Wei, Yunbo Tang, Zongpei Teng, Zexi Peng, Zongxia Li, Chengsong Huang, Yicheng He, et al. A systematic survey of self-evolving agents: From model-centric to environment-driven co-evolution. *TechRxiv*, 2026.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pp. 52040–52094, 2024.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

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. *arXiv preprint arXiv:2505.09388*, 2025.

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

Shuo Yang, Caren Han, Xueqi Ma, Yan Li, Mohammad Reza Ghasemi Madani, and Eduard Hovy. EVOTOOL: Self-evolving tool-use policy optimization in LLM agents via blame-aware mutation and diversity-aware selection. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 43553–43572, 2026b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Xingkun Yin and Hongyang Du. GLOVE: Global verifier for LLM memory-environment realignment. *arXiv preprint arXiv:2601.19249*, 2026.

Hongzhuo Yu, Fei Zhu, Guo-Sen Xie, and Ling Shao. Self-consolidation for self-evolving agents. *arXiv preprint arXiv:2602.01966*, 2026a.

Xinlei Yu, Chengming Xu, Zhangquan Chen, Bo Yin, Cheng Yang, Yongbo He, Yihao Hu, Jiangning Zhang, Cheng Tan, Xiaobin Hu, and Shuicheng Yan. Dual latent memory for visual multi-agent system. In *Forty-third International Conference on Machine Learning*, 2026b.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R. Fung, Hao Peng, and Heng Ji. CRAFT: Customizing LLMs by creating and retrieving from specialized toolsets. In *The Twelfth International Conference on Learning Representations*, 2024.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-R: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*, 2025.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, et al. AgentEvolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*, 2025.

Jiayi Zhang, Yiran Peng, Fanqi Kong, Cheng Yang, Yifan Wu, Zhaoyang Yu, Jinyu Xiang, Jianhao Ruan, Jinlin Wang, Maojia Song, et al. AutoEnv: Automated environments for measuring cross-environment agent learning. *arXiv preprint arXiv:2511.19304*, 2025.

Zhiwei Zhang, Fei Zhao, Rui Wang, Zezhong Wang, Bin Liang, Jiakang Wang, Yao Hu, Shaosheng Cao, and Kam-Fai Wong. Robust tool use via Fission-GRPO: Learning to recover from execution errors. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 40477–40491, 2026.

Zhipu AI. GLM-4.6 model card. Hugging Face model card, 2025. `https://huggingface.co/zai-org/GLM-4.6`.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024.

Adam Zweiger, Jyothish Pari, Han Guo, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 74084–74115, 2025.

A Benchmark and Evaluation Details

BFCL V3. Our in-distribution evaluation uses the multi-turn subset of the Berkeley Function-Calling Leaderboard (BFCL) V3. The benchmark evaluates whether an agent can correctly use executable tools over multi-turn dialogue contexts. Each task contains a user instruction sequence, a set of available tools, and an executable evaluation protocol. The subset contains 800 examples across four categories: *Base*, *Missing Functions*, *Missing Parameters*, and *Long Context*. *Base* evaluates standard multi-turn function calling with complete tool information. *Missing Functions* tests whether the agent can recognize when required tools are unavailable or when no valid tool applies. *Missing Parameters* evaluates whether the agent can handle underspecified arguments rather than hallucinating missing values. *Long Context* stresses state tracking and tool-use consistency over extended dialogue histories.

The 800 examples comprise 200 scenario IDs across the four categories, with 200 examples per category. Following the 400-training / 400-evaluation setup used by Environment Tuning (Lu et al., 2026), we construct our scenario-disjoint split by shuffling the IDs with Python seed 42 and assigning 100 to training and 100 to evaluation across all four categories. This gives 400 examples in each partition, with 100 per category. The final evaluation split is not used for policy training or interface adaptation. The four subsets are benchmark categories, not component ablations. All compared methods use the same split. The task-ID manifests and split checks specify the data independently of the random seed used for policy training.

BFCL V4. For out-of-distribution evaluation, we use the Web Search and Memory tracks from BFCL V4. These tracks differ from BFCL V3 in tool schema structure, task distribution, and interaction pattern. The Web Search track evaluates whether an agent can retrieve and synthesize information using search-style tools. The Memory track evaluates whether an agent can store, retrieve, and update memory-like information across interactions. Since SEAL is trained only on the BFCL V3 training split, BFCL V4 serves as a held-out benchmark for testing cross-distribution transfer.

τ^2-bench. We also evaluate on τ^2-bench (Barres et al., 2025) across the Retail, Airline, and Telecom domains. These tasks require multi-turn tool use in simulated customer-service environments, where the agent must follow domain policies, interact with tools, and maintain task state across turns. Compared with BFCL, τ^2-bench introduces different tool semantics, domain constraints, and user-agent interaction patterns, making it a complementary testbed for OOD generalization.

Evaluation protocol and metric. All evaluations are performed using the original benchmark environments and verifiers. SEAL-specific diagnostic feedback, interface augmentation, capability-specific hints, and recovery-oriented training messages are disabled during evaluation. Thus, all reported results measure the policy under the original test-time protocol. We report success rate as the primary metric:

$$\mathrm{SR} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathbf{1}\left[\mathcal{V}(\tau_q, q, \mathcal{E}) = 1\right], \tag{7}$$

where $\mathcal{Q}$ is the evaluation set and τ_q is the trajectory generated for instruction q. Category-level scores are computed on the corresponding subset, and average scores are computed as the arithmetic mean over reported categories.

B Backbones and Reference Systems

Trained backbones. Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct (Yang et al., 2024) are general-purpose instruction-tuned models from the Qwen2.5 family and serve as the two general backbones. ToolACE-2-Llama-3.1-8B (Liu et al., 2025b) is a Llama-3.1-8B model post-trained on ToolACE function-calling data and serves as the tool-specialized backbone. All three are trained with Vanilla RL and with SEAL under the protocol of Section 4.1.

Method-level baseline. Environment Tuning (Lu et al., 2026) improves multi-turn tool use by changing the training environment rather than supplying expert trajectories: a four-stage curriculum (format and tool-call rewards, then the Base split, then the missing-function, missing-parameter,

and long-context splits, then all scaffolding removed), actionable error augmentation that replaces uninformative execution errors with pedagogical hints, and a per-turn progress reward averaged over the episode, trained with a GRPO variant. We run it on the same backbones and 400-example split as SEAL. It differs from SEAL in changing the reward and using hand-written per-domain hints, whereas SEAL keeps the binary verifier reward and derives its cues from diagnosed failure classes.

Reference systems in Table 2. These rows are reported only to bound the attainable range on BFCL V3; they differ from our backbones in scale, training data, and recipe and are not budget-matched. Closed-source: Gemini-3-Pro-Preview (Google, 2025) and Gemini-2.5-Flash (Comanici et al., 2025) from Google, Claude-Sonnet-4.5 (Anthropic, 2025) from Anthropic, and GPT-4o-2024-11-20 (Hurst et al., 2024) from OpenAI. Open-source: GLM-4.6 (355B) (Zhipu AI, 2025), Qwen3-32B (Yang et al., 2025), Qwen2.5-14B-Instruct (Yang et al., 2024), the larger sibling of our Qwen backbones, and xLAM-2-3b-fc-r (Salesforce AI Research, 2025), a 3B function-calling model trained on large-scale synthetic multi-turn tool-use data, which is why it sits far above our 400-example 3B rows.

Reference systems in Table 3. BitAgent-8B (BitAgent, 2025) and watt-tool-8B (watt-ai, 2024) are open 8B tool-calling models; ToolACE-2-Llama-3.1-8B (Liu et al., 2025b) is our tool-specialized backbone evaluated without training; Qwen3-30B-A3B-Instruct (Yang et al., 2025) is a mixture-of-experts model with about 3B active parameters. None of these rows is budget-matched to our trained models.

C SEAL Training Algorithm

Algorithm 1 specifies the full SEAL training procedure. SEAL alternates between policy rollout, verifier-grounded diagnosis, diagnosis-guided interface evolution, and policy optimization. The interface wrapper is used only during training and is removed during final evaluation. The main Qwen2.5-7B SEAL checkpoint uses both online recovery selection and diagnosis-aware policy weighting. Its correspondence to the reported results is specified in Appendix E.5.

One round is one optimizer iteration with 32 prompts and eight trajectories per prompt. Diagnoses determine the current trajectories' weights and the next round's interface. There is no retroactive change to sampled actions or tool outputs.

At evaluation time, SEAL discards the training-time interface wrapper and evaluates π_{θ_R} in the original environment $\mathcal{E}$ with the unchanged verifier $\mathcal{V}$.

D Theoretical Analysis

This appendix states what the update rules of Section 3 do and do not guarantee. Every statement concerns the rules as specified, at the population or single-batch level; none is a finite-sample or convergence guarantee for the full training loop. Notation follows Section 3: a prompt yields a group of G rollouts with binary verifier rewards $r_j \in \{0, 1\}$ and group-normalized advantages A_j; each trajectory has a diagnostic profile p_j over labels $\mathcal{Z}$; utilities ρ_t follow Eqs. (4)–(5); weights satisfy $w_j \in [w_{\min}, w_{\max}] = [0.5, 2.0]$; and the augmented observation is $\tilde{o}_i = \Omega_t(o_i, \mathcal{F}, H_t, C_{t-1})$.

D.1 Credit Under a Binary Reward

Lemma 1 (Binary GRPO credit is failure-blind)**.** *Consider a group of G rollouts of one prompt with k successes, $0 < k < G$, and advantages $A_j = (r_j - \mu)/\sigma$ with $\mu = k/G$ and $\sigma = \sqrt{\mu(1-\mu)}$. Then every successful trajectory receives $A^+ = \sqrt{(G-k)/k}$ and every failed trajectory receives $A^- = -\sqrt{k/(G-k)}$. Hence the coefficient of $\nabla_\theta \log \pi_\theta(\tau_j)$ in the unclipped reward-surrogate gradient at the rollout policy is the same for all failed trajectories of the group, whatever their failure modes. If $k \in \{0, G\}$, this reward term contributes zero; a separate KL term need not vanish.*

Proof. For $r_j = 1$, $A_j = (1-\mu)/\sqrt{\mu(1-\mu)} = \sqrt{(1-\mu)/\mu} = \sqrt{(G-k)/k}$; for $r_j = 0$, $A_j = -\mu/\sqrt{\mu(1-\mu)} = -\sqrt{k/(G-k)}$. If $k \in \{0, G\}$ then $\sigma = 0$ and GRPO assigns zero

Algorithm 1 SEAL: Synergistic Co-Evolution of Agents and Learning Environments

Require: Initial policy π_{θ_0}, reference policy π_{ref}, training prompts $\mathcal{D}$, base environment $\mathcal{E}$, verifier $\mathcal{V}$, number of rounds R, rollouts per prompt G
Require: Initial interface wrapper Ω_0, base utilities $\rho_0(z)$, clipping range $[w_{\min}, w_{\max}]$
for $r = 0, 1, \ldots, R-1$ **do**
  Sample prompt batch $\mathcal{X}_r \subset \mathcal{D}$
  Construct training environment $\mathcal{E}_{\Omega_r} \leftarrow \Omega_r(\mathcal{E})$
  Initialize rollout buffer $\mathcal{R}_r \leftarrow \emptyset$
  for each prompt $x \in \mathcal{X}_r$ **do**
    Sample G trajectories from π_{θ_r} in $\mathcal{E}_{\Omega_r}$
    for each trajectory τ_j **do**
      Compute verifier reward $r_j \leftarrow \mathcal{V}(\tau_j, x, \mathcal{E})$
      Diagnose trajectory $Z(\tau_j) \leftarrow \Psi(\tau_j, \mathcal{E}, \mathcal{V})$
      Store $(x, \tau_j, r_j, Z(\tau_j))$ in $\mathcal{R}_r$
    end for
  end for
  Aggregate failure profile $C_r \leftarrow \text{Agg}(\{Z(\tau_j) : \tau_j \in \mathcal{R}_r\})$
  Update ρ_r once using Eqs. (4)–(5)
  for each prompt group in $\mathcal{R}_r$ **do**
    Compute group-relative advantages A_j from verifier rewards $\{r_j\}_{j=1}^{G}$
    Compute weights $w_j \leftarrow \text{clip}(\sum_{z \in \mathcal{Z}} p_{\tau_j}(z)\rho_r(z), w_{\min}, w_{\max})$
  end for
  Set weighted advantages $\widetilde{A}_j \leftarrow w_j A_j$ for every trajectory
  Update policy $\pi_{\theta_r} \rightarrow \pi_{\theta_{r+1}}$ using GRPO and KL regularization
  Update interface $\Omega_{r+1} \leftarrow U(\Omega_r, C_r)$
end for
return trained policy π_{θ_R}

advantage to the group. Using the unbiased sample standard deviation instead of σ rescales every advantage of the group by the common factor $\sqrt{(G-1)/G}$, preserving equality within each reward class, but rescaling the displayed values. □

The lemma formalizes challenge ❷ of the introduction: under a terminal binary reward, the scalar group-normalized advantage cannot distinguish an invalid argument from a wrong final answer. An additional weight w_j can distinguish their diagnostic profiles. This statement does not identify the full token-level, clipped, KL-regularized update with a trajectory score-function gradient.

Remark 1 (Division of labor between the two sides). If all binary verifier rewards in a prompt group match, every group-relative reward advantage is zero. Multiplying by w_j therefore leaves the group's reward-driven GRPO policy-gradient contribution at zero; KL and other auxiliary regularizers may still contribute gradients. Reweighting changes the reward signal only in mixed-success groups. The interface Ω_t instead changes observations for subsequent sampling, potentially also for currently unsolved prompts. Updates from other prompts and KL regularization can also change their future rollouts. These are distinct interventions, not a proof that either improves learning or that their effects are independent.

D.2 Utility Dynamics

Proposition 1 (Utility dynamics)**.** *Let $\rho_0(z) \in [0.5, 2.0]$ for all $z \in \mathcal{Z}$ and let ρ_t^* and ρ_t follow Eqs.* (4) *and* (5)*. For every non-pass label z and round $t \geq 1$:*

(a) Invariance. $\rho_t(z) \in [0.5, 2.0]$.

(b) Bounded drift. $|\rho_t(z) - \rho_{t-1}(z)| \leq 0.25\,(2.0 - 0.5) = 0.375$.

(c) Bounded, monotone target. $\rho_0(z) \leq \rho_t^*(z) \leq \min\{2\rho_0(z),\, 2.0\}$*, and $\rho_t^*(z)$ is nondecreasing in the label's share $p_t(z)$, in its co-failure rate $1 - \bar{r}_t(z)$, and in its support $n_t(z)$.*

(d) Geometric tracking. *If* $\rho_s^*(z) = \rho^*(z)$ *for all* $s \in (t_0, t]$, *then* $|\rho_t(z) - \rho^*(z)| \le 0.75^{\,t-t_0}\,|\rho_{t_0}(z) - \rho^*(z)|$*; if instead the target moves by at most* δ *per round, then* $\limsup_t |\rho_t(z) - \rho_t^*(z)| \le 3\delta$.

(e) Self-limiting emphasis. *If* $p_t(z) \to 0$ *or* $\bar{r}_t(z) \to 1$, *then* $\rho_t^*(z) \to \rho_0(z)$ *and* $\rho_t(z) \to \rho_0(z)$.

Proof. (a) By induction: $\rho_t^*(z) \in [0.5, 2.0]$ by the clip, and $\rho_t(z)$ is a convex combination of $\rho_{t-1}(z)$ and $\rho_t^*(z)$. (b) $\rho_t - \rho_{t-1} = 0.25\,(\rho_t^* - \rho_{t-1})$, and both terms lie in $[0.5, 2.0]$. (c) Each of $p_t(z)$, $1-\bar{r}_t(z)$, and $c_t(z)$ lies in $[0, 1]$, so the bracket in Eq. (4) lies in $[\rho_0(z), 2\rho_0(z)]$ and is nondecreasing in each factor; clipping preserves monotonicity, and the lower clip is inactive because $\rho_0(z) \ge 0.5$. (d) Let $e_t = \rho_t(z) - \rho_t^*(z)$. Eq. (5) gives $e_t = 0.75\,\big(e_{t-1} + \rho_{t-1}^*(z) - \rho_t^*(z)\big)$. With a constant target the second term vanishes and $e_t = 0.75\,e_{t-1}$; with $|\rho_{t-1}^*(z) - \rho_t^*(z)| \le \delta$, $|e_t| \le 0.75\,|e_{t-1}| + 0.75\,\delta$, whose fixed point is 3δ. (e) The bracket tends to $\rho_0(z)$, so $\rho_t^*(z) \to \rho_0(z)$, and by (d) the moving average converges to the same limit. □

In words, the emphasis on a failure type rises while that type is frequent and unproductive, can at most double its base value, moves by at most 0.375 per round, closes 90% of the gap to a stationary target within eight rounds ($0.75^8 \approx 0.10$), and returns to the base value once the type disappears from the on-policy rollouts. The last property is what reduces persistent emphasis after a label disappears; it does not prove that the method fixes a failure. The variant observations in Table 7 are not a validation of these dynamics.

D.3 Reweighting as a Bounded Tilt of the Rollout Distribution

Proposition 2 (Bounded tilt)**.** *Let* $w : \tau \mapsto [w_{\min}, w_{\max}]$ *with* $0 < w_{\min} \le w_{\max}$ *and let* $c(\tau)$ *be any coefficient function; both are treated as constants with respect to* θ*, as the GRPO advantage is. Define* $Z_w = \mathbb{E}_{\tau\sim\pi_\theta}[w(\tau)]$ *and the tilted distribution* $q_w(\tau) = w(\tau)\,\pi_\theta(\tau)/Z_w$. *Then*

$$\mathbb{E}_{\tau\sim\pi_\theta}\big[w(\tau)\,c(\tau)\,\nabla_\theta \log \pi_\theta(\tau)\big] = Z_w\,\mathbb{E}_{\tau\sim q_w}\big[c(\tau)\,\nabla_\theta \log \pi_\theta(\tau)\big], \tag{8}$$

and (i) q_w *and* π_θ *have the same support; (ii)* $w_{\min}/w_{\max} \le q_w(\tau)/\pi_\theta(\tau) \le w_{\max}/w_{\min}$ *for every* τ*; (iii)* $\mathrm{KL}(q_w \,\|\, \pi_\theta) \le \log(w_{\max}/w_{\min})$ *and* $\mathrm{KL}(\pi_\theta \,\|\, q_w) \le \log(w_{\max}/w_{\min})$.

Proof. Eq. (8) follows from $q_w(\tau) = w(\tau)\pi_\theta(\tau)/Z_w$ by substitution. (i) $w > 0$. (ii) $q_w/\pi_\theta = w/Z_w$ and $w_{\min} \le Z_w \le w_{\max}$. (iii) $\mathrm{KL}(q_w\|\pi_\theta) = \mathbb{E}_{q_w}[\log(w/Z_w)] \le \log(w_{\max}/w_{\min})$ by (ii), and likewise $\mathrm{KL}(\pi_\theta\|q_w) = \mathbb{E}_{\pi_\theta}[\log(Z_w/w)] \le \log(w_{\max}/w_{\min})$. □

With SEAL's range $[0.5, 2.0]$, the density ratio lies in $[1/4, 4]$ and both divergences are at most $\log 4 \approx 1.39$ nats. The weighted score-function term can therefore be represented under a distribution that emphasizes high-utility trajectories, with prefactor $Z_w \in [0.5, 2.0]$. This algebra neither changes the sampling procedure nor equates the full clipped, KL-regularized GRPO objective to an unweighted objective. Positive weights preserve each advantage's sign. The corresponding finite-batch identity is $\sum_j w_j A_j \nabla_\theta \log \pi_\theta(\tau_j) = W \sum_j (w_j/W)\,A_j \nabla_\theta \log \pi_\theta(\tau_j)$ with $W = \sum_j w_j \in [N w_{\min}, N w_{\max}]$. The prefactor is bounded, but the ratio of weighted to unweighted aggregate gradient norms is not: unequal weights can change cancellation between trajectory gradients.

Corollary 1 (The utility-average clip never binds)**.** *Since* p_j *is a probability distribution over labels and* $\rho_t(z) \in [0.5, 2.0]$ *for all* z *(Proposition 1(a)),* $\sum_z p_j(z)\rho_t(z)$ *is a convex combination of utilities and lies in* $[\min_z \rho_t(z), \max_z \rho_t(z)] \subseteq [0.5, 2.0]$. *The clip in the definition of* w_j *is therefore inactive. The direct multiplier remains in* $[0.5, 2.0]$*, so the density-ratio bound of 4 in Proposition 2(ii) applies.*

D.4 Interface Evolution: Policy Preservation and a Finite Channel

Proposition 3 (Interface evolution)**.** *Let* $\tilde{o}_i = o_i \oplus \phi_{\mathrm{schema}}(\mathcal{F}) \oplus \phi_{\mathrm{err}}(o_i, H_t) \oplus \phi_{\mathrm{cap}}(C_{t-1})$*, where* ϕ_{schema} *is a deterministic function of the tool schema, and* ϕ_{err} *and* ϕ_{cap} *instantiate fixed templates chosen, as a function of* (H_t, C_{t-1})*, from a finite set* $\mathcal{L}$ *of complete template configurations, including cue presence, ordering, and placement within one round; the transition dynamics, the tool outputs, and the reward* $r(\tau) = \mathcal{V}(\tau, q, \mathcal{E})$ *are unchanged. Then:*

(i) Policy preservation. *For every unrestricted policy π over original observations, the policy $\tilde{\pi}(a \mid \tilde{o}_{\le i}) = \pi(a \mid o_{\le i})$ over augmented observations induces the same trajectory distribution; hence $\sup_{\tilde{\pi}} J(\tilde{\pi}) \ge \sup_{\pi} J(\pi)$, and the set of verifier-accepted trajectories is identical with and without the interface.*

(ii) Finite-channel bound. *Let R_q be the reference of task q (its reference call sequence, hidden parameter values, and final answer), and let $\ell \in \mathcal{L}$ be the template configuration selected in round t. Conditionally on the agent's own observation history $o_{\le i}$ and the schema $\mathcal{F}$, the augmented observations depend on R_q (through the verifier-derived labels that shape H_t and C_{t-1}) only via ℓ, so $I(R_q; \tilde{o}_{\le i} \mid o_{\le i}, \mathcal{F}) \le H(\ell \mid o_{\le i}, \mathcal{F}) \le \log_2 |\mathcal{L}|$ bits per round.*

Proof. (i) o_i is the first component of $\tilde{o}_i$, so $\tilde{\pi}$ is well defined and reproduces the action distribution of π at every turn; the transition kernel and the verifier do not read $\tilde{o}_i$, so the induced trajectory distribution and its return coincide, which gives the inequality after taking suprema. Acceptance depends only on $(\tau, q, \mathcal{E})$. (ii) Given $(o_{\le i}, \mathcal{F})$, $\tilde{o}_{\le i}$ is a deterministic function of ℓ; the data-processing inequality gives $I(R_q; \tilde{o}_{\le i} \mid o_{\le i}, \mathcal{F}) \le I(R_q; \ell \mid o_{\le i}, \mathcal{F}) \le H(\ell \mid o_{\le i}, \mathcal{F}) \le \log_2 |\mathcal{L}|$. □

Part (i) concerns unrestricted policies and append-only observations; it is not a guarantee for a fixed-capacity model or a context window that truncates original observations. Part (ii) is a conditional, single-round bound. The cardinality must count complete configurations, not just label names or individual strings. Even one bit can reveal an answer among two candidates, and information may accumulate across rounds; the bound is not a no-leakage guarantee. A transcript of T configurations has at most $T \log_2 |\mathcal{L}|$ bits of entropy under the same finite-configuration assumption. Neither part guarantees performance after removing training cues. The removed-at-evaluation results are empirical observations, not a consequence of these propositions.

E Experimental Details

This section provides implementation details for reproducing the training runs and the controlled Vanilla RL comparison. Within each backbone, the binary-reward comparison uses the same training split, rollout budget, decoding configuration, and evaluation verifier. Environment Tuning follows its released curriculum, reward design, and training recipe and serves as a method-level baseline. Vanilla RL provides the matched binary-reward control under the same GRPO backbone and rollout protocol.

E.1 Hyperparameter Configuration

Table 5 summarizes the main training configuration used in our BFCL experiments. We use Qwen2.5-7B-Instruct as the default backbone for the main BFCL experiments and apply the same GRPO configuration to Vanilla RL and SEAL. The only algorithmic differences are whether verifier-grounded diagnosis, diagnosis-guided interface evolution, closed-loop failure-profile updates, and diagnostic advantage weighting are enabled.

For comparisons with different backbones, we keep the same data split, rollout budget, verifier, decoding configuration, and evaluation protocol whenever possible. When hardware constraints require changes to micro-batch size or gradient accumulation, we keep the effective batch size fixed.

E.2 Diagnosis Label Priority

Table 1 in the main text specifies the operational decision sequence together with the fixed base utilities ρ_0. These seven outcomes are the labels used for training; the generic behavioral terms in Figure 2 are not a second taxonomy. An exception while executing a reference program or running a checker produces `gt_error` or `checker_error`; these bookkeeping labels are excluded from the learning utilities. Required tool use and acceptance are defined by the original executable verifier, and reference information stays inside the checker.

This ordering separates absence of a required call from state and response failures. It does not assert that every invalid argument has a unique diagnostic label, or that every downstream mismatch is a new causal error. Model-side and reference-side execution use separate state instances.

Table 5: **Main hyperparameter configuration for BFCL experiments.** Unless otherwise specified, Vanilla RL and SEAL use the same optimization, rollout, and evaluation settings.

Category	Hyperparameter	Value
Policy optimization	Base model	Qwen2.5-7B-Instruct
	Training method	GRPO
	Actor learning rate	1×10^{-6}
	Train batch size	32
	PPO mini-batch size	8
	PPO micro-batch size / GPU	1
Rollout	Inference engine	vLLM (async)
	Rollouts per prompt	8
	Training temperature	1.0
	Validation temperature	0.0
	Max prompt length	8192
	Max response length	4096
	Max model length	16384
Multi-turn setting	Number of training samples	400
	Multi-turn rollout	Enabled
	Max interaction steps	20
	Tool format	`llama3_json`
	BFCL tool prompt mode	`bfcl_qwen_fc`
Regularization	Reward KL penalty	Disabled
	Actor KL loss coefficient	0.001
	Entropy coefficient	0.0
Hardware	GPUs	4
	Tensor parallel size	1
	GPU memory utilization	0.6
	Max environment workers	32

The trajectory weight averages valid diagnostic utilities and directly scales the normalized group-relative advantage, without an additional outcome-calibration stage or subsequent advantage renormalization. The same trajectory-level multiplier applies to positive and negative advantages and to all response tokens. Base utilities, EMA coefficient 0.25, and support constant 32 are fixed across the matched experiments. For example, two correct abstentions and one state mismatch give $w = (1.8 + 1.8 + 1.0)/3 \approx 1.53$ under the base utilities. This trajectory-level form preserves the underlying GRPO objective's token-level loss structure.

E.3 Controlled Comparison Protocol

The comparison between Vanilla RL and SEAL is controlled along the following dimensions. Both methods use the same initial checkpoint, the same set of 400 BFCL V3 training samples, the same number of rollouts per prompt, the same optimizer, and the same decoding configuration. Both methods are evaluated using the original BFCL V3 multi-turn environment and the official verifier. These matches concern examples, rollouts, and optimization settings; wall-clock time and prompt-token exposure can differ because SEAL adds training-time interface content.

Vanilla RL optimizes the standard GRPO objective using only verifier rewards. SEAL uses the same verifier rewards and the same GRPO backbone, but additionally uses verifier-grounded diagnoses from training rollouts to construct training-time interface updates and diagnostic utility weights. No policy receives ground-truth tool trajectories, hidden parameter values, reference intermediate actions, or final answers as training observations; the executable verifier internally uses benchmark references to compute outcomes.

The evolved SEAL interface is used only during training. During evaluation, the wrapper is removed, and the model interacts with the original benchmark interface. Thus, SEAL and Vanilla

RL are evaluated under identical test-time conditions. The comparison therefore measures whether the policy has internalized better tool-use behavior, rather than whether it has access to additional test-time information.

E.4 Logging and Reproducibility

For each reported training run, we record the random seed, base checkpoint, code version, training configuration and overrides, rollout budget, decoding parameters, round-level diagnostic profiles, interface states, validation scores, and final per-example evaluation outputs. These records provide run-level correspondence between the training configuration, checkpoint, and reported evaluation result.

E.5 Run and Configuration Correspondence

Main Qwen2.5-7B checkpoint. The Qwen2.5-7B SEAL result in Table 2 is the full-online configuration specified in Algorithm 1. During training, it uses fixed required-argument and enum schema affordances, online diagnosis-conditioned recovery selection, and diagnosis-aware advantage reweighting.

At round t, trajectories are collected under the current interface Ω_t. Their verifier-grounded diagnoses determine the advantage weights for the current policy update and the recovery guidance selected for Ω_{t+1}. Schema affordances remain fixed throughout training, while recovery guidance evolves from preceding on-policy diagnoses.

The resulting checkpoint scores 161/400 (40.25%) on the BFCL V3 evaluation split, with Base, Missing Functions, Missing Parameters, and Long Context accuracies of 58.00%, 36.00%, 34.00%, and 33.00%, respectively. The Qwen2.5-7B SEAL OOD results in Table 3 evaluate this same checkpoint without additional training or checkpoint selection. All SEAL-specific interface additions are removed during evaluation, and the checkpoint is evaluated with the original BFCL environment and verifier. Factorial control configurations are specified in Appendix H.1.

F Prompt and Interface Templates

This section reports the prompt and training-time interface templates used in our BFCL implementation. The base tool-use prompt is shared by Vanilla RL and SEAL. SEAL modifies only the training-time environment interface, for example by annotating tool schemas, enriching failed tool feedback when enabled, or injecting short behavioral guidelines derived from recent diagnostic profiles. All SEAL-specific interface augmentations are disabled during evaluation.

F.1 Base BFCL Tool-Use Prompt

For Qwen-style BFCL experiments, the environment serializes the available tools into the system prompt using the following template. This base prompt is used by both Vanilla RL and SEAL.

BFCL Qwen Tool Prompt Template

```
# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools>
    XML tags:
<tools>
$tool_schema_json_1
$tool_schema_json_2
...
</tools>
```

```
For each function call, return a json object with function name
    and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>
```

The base prompt defines the tool-call format and exposes the available tool schemas. It does not contain SEAL-specific diagnostic labels, recovery hints, or capability-dependent instructions.

F.2 Schema Observation Annotation

In our active BFCL configuration, SEAL uses a lightweight observation annotation mechanism. When `observation_lite` is enabled, SEAL does not change tool APIs, tool signatures, tool outputs, or verifier behavior. Instead, it augments the exposed schema descriptions with lightweight annotations that make existing constraints more explicit.

Schema Observation Annotation Template

```
$argument_description [required]

$argument_description Allowed values: $enum_value_1,
    $enum_value_2, ...
```

The environment also appends the following training-time instruction:

Observation-Lite Instruction

```
Observation note: parameters marked [required] must be provided
    exactly when calling a tool; do not invent tools outside the
    current schema.
```

This annotation corresponds to the schema-affordance component ϕ_{schema}. It exposes only public schema information, such as required arguments and enum constraints, and does not reveal the correct tool choice, hidden parameter values, reference tool trajectory, or final answer for any instance.

F.3 Tool Feedback Evolution Template

When `tool_feedback_evolution` is enabled, SEAL enriches only failed interactions. This component corresponds to the recovery-feedback component ϕ_{err}. It converts low-level parse or execution errors into structured recovery hints, while preserving the original tool API and verifier.

For parser failures, the environment may append format-specific hints such as:

Parse Error Hint Template

```
$base_parse_error

[Hint] Each tool call MUST be a JSON object inside
    <tool_call>...</tool_call> tags with both "name" and
    "arguments" keys, e.g. <tool_call>{"name": "<tool>",
    "arguments": {"<arg>": <value>}}</tool_call>.
[Hint] Available tools: $tool_name_1, $tool_name_2, ...
[Hint] If no available tool can serve the user's request, reply in
    plain text instead of producing a malformed tool_call.
```

For execution errors, the environment may append tool-specific but non-solution-revealing hints such as:

Tool Error Hint Template

```
[ToolHint] Required argument(s) $args for `$tool_name` were
    missing. Re-emit the tool_call with these keys present in
    `arguments`.

[ToolHint] `$tool_name` does not accept argument `$arg`. Drop it
    and only use arguments listed in the tool schema.

[ToolHint] `$tool_name` rejected an argument type. Re-check
    argument types in the tool schema (string vs int vs list).
```

These hints identify the class of tool-use error and the violated interface constraint. They do not provide the next correct tool call, instance-specific argument values, or the ground-truth solution trajectory.

F.4 Diagnostic Evolution Guideline Template

When `diagnostic_evolution` is enabled, SEAL reads recent failure statistics from the capability profile and injects short behavioral guidelines into the system prompt. This component corresponds to ϕ_{cap}: the interface is adjusted according to recurring failure modes of the current policy rather than individual benchmark instances.

Diagnostic-Driven System Guideline Template

```
# Behavioral guidelines (env-evolved from recent failures):
- [Cautious-call] If no available function clearly matches the
    user's request, respond in plain text rather than fabricating a
    tool call.
- [Active-respond] When the user's request CAN be served by an
    available function, you MUST emit a tool_call rather than only
    replying in plain text.
- [State-aware] Before issuing a state-mutating call, re-read the
    most recent tool results to confirm the current state matches
    what the arguments assume.
- [Scope] Verify that the instance/target your call refers to
    actually appeared in earlier tool results before calling on it.
- [Format] Match your final reply to the format implied by the
    user's request: lists, units, exact strings, ordering.
```

The guidelines are selected from aggregate failure patterns, such as spurious tool calls, missing tool calls, state mismatches, or execution-response mismatches. They are capability-level instructions rather than per-instance hints. Their text contains no explicit reference trajectory, hidden state, hidden argument value, or target answer.

Full SEAL and factorial controls. Full SEAL combines static required-argument/enum schema annotations, online diagnosis-conditioned recovery selection, and diagnostic GRPO advantage weighting. Schema annotations remain fixed while recovery guidance is selected from preceding on-policy diagnoses for subsequent rounds. Table 4 crosses a fixed enriched interface or dynamic recovery selection with uniform failure weighting or diagnosis-aware weighting. Static+Uniform, Dynamic+Uniform, Static+Diagnosis-aware, and Full SEAL form the four cells; their shared protocol and weighting definitions are in Appendix H.1.

Concrete recovery example. Suppose an engine-start attempt returns that the brake must be pressed. If the executable state check assigns `state_mismatch`, the next round may expose the following fixed class-level instruction:

> Resolve the stated prerequisite before retrying, and reuse exact values, identifiers, paths, and state returned by tools rather than guessing.

The label is determined by the executable checks, not simply a keyword in the error message. The instruction reveals neither a reference call nor an instance-specific value.

Taxonomy and template provenance. The taxonomy operationalizes BFCL's executable outcomes. The comparison protocol fixes the seven labels, decision order, base utilities, and generic recovery library before launching the matched runs, and keeps them unchanged across conditions and backbones. Rule development and checkpoint selection use a separate development set, disjoint from the 400-example BFCL V3 evaluation split. The evaluation split is reserved for final evaluation and is not used to design or revise the labels, utilities, or recovery templates. The recovery selector receives categorical diagnoses and permitted observable evidence; benchmark references used internally by the verifier are not exposed to the policy or as recovery instructions. All adaptive guidance is removed at evaluation.

G Qualitative Case Studies

We present four qualitative case studies to illustrate tool-use behavior after SEAL training across different capability types: file-system state tracking, API-parameter recovery, cross-domain tool composition, and vehicle-control state maintenance. Each case compares the behavior of the base model before SEAL training with the behavior of the SEAL-trained model on the same BFCL task. We also include the expected outcome for post-hoc analysis only; these references are not exposed to the policy during training or evaluation.

We focus on critical turns where the base model fails and the SEAL-trained model succeeds. These examples are not intended to replace the controlled Vanilla RL comparison in the main experiments; rather, they illustrate the types of tool-use failures that SEAL is designed to diagnose and correct.

G.1 File-System State Tracking

This case studies a file-system task requiring the agent to compare report files, move and rename a note file inside an archive directory, retrieve a support ticket, and finally resolve it. The key challenge is maintaining the current directory state and respecting file-system tool constraints.

Task: BFCL `multi_turn_base_24` **Model:** Base model **Result:** reward = 0.0

User Query 1: **Compare the draft and final report files.**

```
Assistant:
<tool_call>
{"name": "find", "arguments": {"path": ".", "name": "draft_report
    "}}
</tool_call>
<tool_call>
{"name": "find", "arguments": {"path": ".", "name": "final_report
    "}}
</tool_call>
<tool_call>
{"name": "diff", "arguments": {"file_name1": "draft_report",
 "file_name2": "final_report"}}
</tool_call>

Tool Response:
{"matches": []}
{"matches": []}
{"error": "diff: draft_report or final_report: No such file or
    directory"}
```

User Query 2: **Transfer `temp_notes.txt` into `archives` and rename it to `notes_2024.txt`.**

```
Assistant:
<tool_call>
{"name": "mv", "arguments": {"source": "temp_notes.txt",
 "destination": "archives/temp_notes.txt"}}
</tool_call>
<tool_call>
{"name": "mv", "arguments": {"source": "archives/temp_notes.txt",
 "destination": "notes_2024.txt"}}
</tool_call>

Tool Response:
{"error": "mv: path not allowed in destination. Provide only a
    file or directory name."}
{"error": "mv: cannot move 'archives/temp_notes.txt': No such
    file or directory"}

Assistant:
<tool_call>
{"name": "ls", "arguments": {}}
</tool_call>

Tool Response:
{"current_directory_content": ["report_draft.txt", "report_final.
    txt",
 "temp_notes.txt", "archives"]}

Assistant:
<tool_call>
{"name": "mv", "arguments": {"source": "temp_notes.txt",
 "destination": "archives/temp_notes.txt"}}
</tool_call>

Tool Response:
{"error": "mv: path not allowed in destination. Provide only a
    file or directory name."}
```

User Query 3: **Retrieve support ticket 987654.**

```
Assistant:
<tool_call>
{"name": "get_ticket", "arguments": {"ticket_id": 987654}}
</tool_call>

Tool Response:
{"id": 987654, "status": "open",
 "description": "Issue with workstation not booting properly.", "
    resolution": ""}
```

Failure analysis. *The base model locates the relevant file names only after an initial failure, but continues to issue invalid path-style arguments to* `mv`*. This indicates that it does not convert the execution error into a revised state-aware plan.*

Task: BFCL `multi_turn_base_24` **Model:** Base model + SEAL **Result:** reward = 1.0

User Query 1: **Compare the draft and final report files.**

```
Assistant:
<tool_call>
{"name": "find", "arguments": {"path": ".", "name": "draft report
    "}}
</tool_call>
<tool_call>
{"name": "find", "arguments": {"path": ".", "name": "final report
    "}}
</tool_call>

Tool Response:
{"matches": []}
{"matches": []}

Assistant:
<tool_call>
{"name": "ls", "arguments": {}}
</tool_call>

Tool Response:
{"current_directory_content": ["report_draft.txt", "report_final.
    txt",
 "temp_notes.txt", "archives"]}

Assistant:
<tool_call>
{"name": "diff", "arguments": {"file_name1": "report_draft.txt",
 "file_name2": "report_final.txt"}}
</tool_call>

Tool Response:
{"diff_lines": "- Initial draft content for the report.\n+
    Finalized content for the report."}
```

User Query 2: **Transfer `temp_notes.txt` into `archives` and rename it to `notes_2024.txt`.**

```
Assistant:
<tool_call>
{"name": "mv", "arguments": {"source": "temp_notes.txt",
 "destination": "archives"}}
</tool_call>

Tool Response:
{"result": "'temp_notes.txt' moved to 'archives/temp_notes.txt'"}

Assistant:
<tool_call>
{"name": "cd", "arguments": {"folder": "archives"}}
</tool_call>
<tool_call>
{"name": "mv", "arguments": {"source": "temp_notes.txt",
 "destination": "notes_2024.txt"}}
</tool_call>

Tool Response:
{"current_working_directory": "/archives"}
{"result": "'temp_notes.txt' moved to 'notes_2024.txt'"}
```

User Query 3: **Retrieve support ticket 987654.**

```
Assistant:
<tool_call>
{"name": "get_ticket", "arguments": {"ticket_id": 987654}}
</tool_call>

Tool Response:
{"id": 987654, "status": "open",
 "description": "Issue with workstation not booting properly.", "
    resolution": ""}
```

User Query 4: **Resolve the ticket with: `Fixed through manual troubleshooting techniques.`**

```
Assistant:
<tool_call>
{"name": "resolve_ticket", "arguments": {"ticket_id": 987654,
 "resolution": "Fixed through manual troubleshooting techniques
    ."}}
</tool_call>

Tool Response:
{"status": "Ticket 987654 has been resolved successfully."}
```

Success analysis. *The SEAL-trained model uses observable directory contents to infer the actual file names, then respects the local-directory constraint by moving into* `archives` *before renaming the file. This illustrates recovery from tool feedback and file-system state tracking after SEAL training.*

Expected Outcome: Compare `report_draft.txt` and `report_final.txt`; move `temp_notes.txt` into `archives`; rename it to `notes_2024.txt`; retrieve ticket 987654; resolve it with the provided resolution string.

G.2 Recovering from API Parameter Mismatch

This case studies a travel API task. The user asks for the economy flight cost from Rivermist to Stonebrook on October 6, 2026, and later asks to set a spending budget provided in GBP. The model must recover from invalid airport parameters and convert the budget into the expected currency.

Task: BFCL `multi_turn_miss_param_150` **Model:** Base model **Result:** reward = 0.0

User Query 1: **Find the economy flight cost from Rivermist to Stonebrook on October 6, 2026.**

```
Assistant:
<tool_call>
{"name": "get_flight_cost", "arguments": {"travel_from": "RVM",
 "travel_to": "STB", "travel_date": "2026-10-06",
 "travel_class": "economy"}}
</tool_call>

Tool Response:
Error during execution: No available route for the given airports
    .

Assistant:
There is no direct flight route available between Rivermist and
    Stonebrook.
```

User Query 2: **Let's anchor my daily spend at a specific amount.**

```
Assistant:
Please provide authentication details before I can set a budget.
```

User Query 3: **It is 15400 GBP. My access token is `abc123token`.**

```
Assistant:
<tool_call>
{"name": "set_budget_limit", "arguments": {"access_token": "
    abc123token",
 "budget_limit": 15400.0}}
</tool_call>

Tool Response:
{"budget_limit": 15400.0}
```

Failure analysis. *The base model treats guessed airport codes as final and does not use the airport-resolution tools after the route error. It also sets the GBP amount directly as the budget limit, missing the required currency conversion.*

Task: BFCL `multi_turn_miss_param_150` **Model:** Base model + SEAL **Result:** reward = 1.0

User Query 1: **Find the economy flight cost from Rivermist to Stonebrook on October 6, 2026.**

```
Assistant:
<tool_call>
{"name": "get_flight_cost", "arguments": {"travel_from": "RVM",
 "travel_to": "STB", "travel_date": "2026-10-06",
 "travel_class": "economy"}}
</tool_call>

Tool Response:
Error during execution: No available route for the given airports
    .

Assistant:
<tool_call>
{"name": "get_nearest_airport_by_city",
 "arguments": {"location": "Rivermist"}}
</tool_call>
<tool_call>
{"name": "get_nearest_airport_by_city",
 "arguments": {"location": "Stonebrook"}}
</tool_call>

Tool Response:
{"nearest_airport": "RMS"}
{"nearest_airport": "SBK"}

Assistant:
<tool_call>
{"name": "get_flight_cost", "arguments": {"travel_from": "RMS",
 "travel_to": "SBK", "travel_date": "2026-10-06",
 "travel_class": "economy"}}
</tool_call>

Tool Response:
{"travel_cost_list": [220.0]}
```

User Query 3: **It is 15400 GBP. My access token is `abc123token`.**

```
Assistant:
<tool_call>
{"name": "compute_exchange_rate", "arguments": {"base_currency":
    "GBP",
 "target_currency": "USD", "value": 15400}}
</tool_call>

Tool Response:
{"exchanged_value": 22000.0}

Assistant:
<tool_call>
{"name": "set_budget_limit", "arguments": {"access_token": "
    abc123token",
 "budget_limit": 22000.0}}
</tool_call>

Tool Response:
{"budget_limit": 22000.0}
```

Success analysis. *The SEAL-trained model recovers by using the missing airport-resolution tools and later converts GBP to USD before calling* `set_budget_limit`*. This illustrates successful handling of intermediate API dependencies and parameter constraints.*

Expected Outcome: Resolve Rivermist to `RMS` and Stonebrook to `SBK`; query `get_flight_cost(RMS, SBK, 2026-10-06, economy)`; convert 15400 GBP to 22000 USD; set the budget limit to 22000.0.

G.3 Cross-Domain Tool Composition

This case combines stock-market and Twitter APIs. The model must retrieve the price of Omega Industries, insert the price into a tweet, include `#TechBoom` and `@industryexperts`, and later add `@technewsworld` to the existing tweet.

Task: BFCL `multi_turn_base_147` **Model:** Base model **Result:** reward = 0.0

User Query 1: **Find Omega Industries' price and post a tweet with the price, `#TechBoom`, and `@industryexperts`.**

```
Assistant:
<tool_call>
{"name": "get_stock_info", "arguments": {"symbol": "OMGA"}}
</tool_call>

Tool Response:
{"error": "Stock with symbol 'OMGA' not found."}

Assistant:
Let's use a more common tech stock, such as AAPL.

<tool_call>
{"name": "get_stock_info", "arguments": {"symbol": "AAPL"}}
</tool_call>

Tool Response:
{"price": 227.16, ...}

Assistant:
<tool_call>
{"name": "post_tweet", "arguments": {"content":
 "Apple Inc. is skyrocketing at $227.16 per share! ...",
 "tags": ["#TechBoom"], "mentions": ["@industryexperts"]}}
</tool_call>
```

User Query 2: **Add `@technewsworld` to amplify the message.**

```
Assistant:
<tool_call>
{"name": "post_tweet", "arguments": {"content":
 "Apple Inc. is skyrocketing at $227.16 per share! ...",
 "tags": ["#TechBoom"],
 "mentions": ["@industryexperts", "@technewsworld"]}}
</tool_call>
```

Failure analysis. *The base model substitutes Apple for Omega Industries after the guessed symbol fails, thereby changing the user's target entity. It also posts a new tweet rather than adding the new mention to the existing tweet.*

Task: BFCL `multi_turn_base_147` **Model:** Base model + SEAL **Result:** reward = 1.0

User Query 1: **Find Omega Industries' price and post a tweet with the price, `#TechBoom`, and `@industryexperts`.**

```
Assistant:
<tool_call>
{"name": "get_stock_info", "arguments": {"symbol": "OMGA"}}
</tool_call>

Tool Response:
{"error": "Stock with symbol 'OMGA' not found."}

Assistant:
<tool_call>
{"name": "get_symbol_by_name",
 "arguments": {"name": "Omega Industries"}}
</tool_call>

Tool Response:
{"symbol": "OMEG"}

Assistant:
<tool_call>
{"name": "get_stock_info", "arguments": {"symbol": "OMEG"}}
</tool_call>

Tool Response:
{"price": 457.23, ...}

Assistant:
<tool_call>
{"name": "post_tweet", "arguments": {"content":
 "Omega Industries is skyrocketing at $457.23 per share! ...",
 "tags": ["#TechBoom"], "mentions": ["@industryexperts"]}}
</tool_call>
```

User Query 2: **Add `@technewsworld` to amplify the message.**

```
Assistant:
<tool_call>
{"name": "mention", "arguments": {"tweet_id": 1,
 "mentioned_usernames": ["@technewsworld"]}}
</tool_call>

Tool Response:
{"mention_status": "Users mentioned successfully"}
```

Success analysis. *The SEAL-trained model treats the failed symbol lookup as recoverable evidence, resolves the company name to* `OMEG`*, retrieves the correct price, and updates the existing tweet with the dedicated* `mention` *tool.*

Expected Outcome: Resolve Omega Industries to `OMEG`; retrieve price 457.23; post a tweet about Omega Industries with `#TechBoom` and `@industryexperts`; add `@technewsworld` to the existing tweet using `mention`.

G.4 Vehicle-Control State Maintenance

This case studies long-context vehicle control. The user first fills the car with 13.2 gallons, asks the agent to fill the tank to its maximum limit, lock the doors, and set the parking brake. Later, the user asks the agent to start the engine and check the fuel level before posting a road-trip tweet.

Task: BFCL `multi_turn_long_context_88` **Model:** Base model **Result:** reward = 0.0

User Query 1: **I filled my car with 13.2 gallons. How much is that in liters?**

```
Assistant:
<tool_call>
{"name": "liter_to_gallon", "arguments": {"gallon": 13.2}}
</tool_call>

Tool Response:
Error during execution: unexpected keyword argument 'gallon'

Assistant:
<tool_call>
{"name": "gallon_to_liter", "arguments": {"gallon": 13.2}}
</tool_call>

Tool Response:
{"liter": 49.967411999999996}
```

User Query 2: **Fill the tank to max, lock all doors, and set the parking brake.**

```
Assistant:
The tank needs 36.8 more gallons.

<tool_call>
{"name": "liter_to_gallon", "arguments": {"liter": 139.32}}
</tool_call>
<tool_call>
{"name": "lockDoors", "arguments": {"unlock": false,
 "door": ["driver", "passenger", "rear_left", "rear_right"]}}
</tool_call>
<tool_call>
{"name": "activateParkingBrake", "arguments": {"mode": "engage"}}
</tool_call>
```

User Query 3: **Start the engine in START mode, check fuel level, and post a road-trip tweet.**

```
Assistant:
<tool_call>
{"name": "startEngine", "arguments": {"ignitionMode": "START"}}
</tool_call>

Tool Response:
{"error": "Brake pedal needs to be pressed when starting the
    engine."}

Assistant:
<tool_call>
{"name": "pressBrakePedal", "arguments": {"pedalPosition": 1}}
</tool_call>
<tool_call>
{"name": "startEngine", "arguments": {"ignitionMode": "START"}}
</tool_call>

Tool Response:
{"engineState": "running", "fuelLevel": 13.2, "batteryVoltage":
    12.6}
```

Failure analysis. *The base model verbally computes the remaining capacity but never executes* `fillFuelTank`*. The later engine check therefore reports* `fuelLevel = 13.2`*, revealing that the required state update did not occur.*

Task: BFCL `multi_turn_long_context_88` **Model:** Base model + SEAL **Result:** reward = 1.0

User Query 1: **I filled my car with 13.2 gallons. How much is that in liters?**

```
Assistant:
<tool_call>
{"name": "gallon_to_liter", "arguments": {"gallon": 13.2}}
</tool_call>

Tool Response:
{"liter": 49.967411999999996}
```

User Query 2: **Fill the tank to max, lock all doors, and set the parking brake.**

```
Assistant:
<tool_call>
{"name": "fillFuelTank", "arguments": {"fuelAmount": 36.8}}
</tool_call>
<tool_call>
{"name": "lockDoors", "arguments": {"unlock": false,
 "door": ["driver", "passenger", "rear_left", "rear_right"]}}
</tool_call>
<tool_call>
{"name": "activateParkingBrake", "arguments": {"mode": "engage"}}
</tool_call>

Tool Response:
{"fuelLevel": 50.0}
{"lockStatus": "locked", "remainingUnlockedDoors": 0}
{"parkingBrakeStatus": "engaged", ...}
```

User Query 3: Start the engine in START mode, check fuel level, and post a road-trip tweet.

```
Assistant:
<tool_call>
{"name": "startEngine", "arguments": {"ignitionMode": "START"}}
</tool_call>

Tool Response:
{"error": "Brake pedal needs to be pressed when starting the
    engine."}

Assistant:
<tool_call>
{"name": "pressBrakePedal", "arguments": {"pedalPosition": 1}}
</tool_call>
<tool_call>
{"name": "startEngine", "arguments": {"ignitionMode": "START"}}
</tool_call>

Tool Response:
{"engineState": "running", "fuelLevel": 50.0, "batteryVoltage":
    12.6}

Assistant:
<tool_call>
{"name": "post_tweet", "arguments": {"content":
 "Embarking on an exciting road trip from SF to Rivermist!",
 "tags": ["#RoadTrip", "#Adventure", "#Exploring"]}}
</tool_call>
```

Success analysis. *The SEAL-trained model executes the state-changing `fillFuelTank` call before later checking the engine state. The observed `fuelLevel = 50.0` is consistent with successful preservation of the long-range state dependency.*

Expected Outcome: Convert 13.2 gallons to approximately 49.97 liters; fill the tank to 50.0 gallons; lock all doors; engage the parking brake; press the brake before starting the engine;

observe `fuelLevel = 50.0`; post the road-trip tweet with `#RoadTrip`, `#Adventure`, and `#Exploring`.

G.5 OOD Qualitative Example: Mobile Data Troubleshooting on τ^2-Bench

Task: τ^2-bench Telecom Mobile Data Troubleshooting
Model: Vanilla Qwen2.5-3B
Result: reward = 0.0

User Query: **My mobile data either stops working or is very slow. I need excellent internet speed on my phone. I am currently abroad in France and do not have access to Wi-Fi.**

```
Assistant:
Could you provide your phone number?

User:
My phone number is 555-123-2002. I am abroad in France.

Assistant:
check_network_status

Tool/User Device:
Airplane Mode: OFF
SIM Card Status: active
Cellular Signal: excellent
Cellular Network Type: 5G
Mobile Data Enabled: Yes
Data Roaming Enabled: No
Wi-Fi Connected: No

Assistant:
Data roaming is off. Please enable data roaming.

User/Device:
Data Roaming is now ON.
Status Bar: Excellent | 5G | Data Enabled | Data Saver

Assistant:
run_speed_test

Tool/User Device:
Speed Test Result: 55.00 Mbps (Good).

User:
The result is good, but I want excellent speed. What should I do
    next?

Assistant:
Try checking data usage, resetting APN settings, or escalating to
     carrier
support.

User:
Please proceed with escalating this issue to carrier support.
###TRANSFER###
```

Note (Failure Point): *The baseline resolves only one issue: data roaming. However, the status bar after enabling roaming still shows* `Data Saver`*. Since the speed test returns only* Good*, the agent should continue diagnosis and check data restrictions. Instead, it drifts toward unrelated steps and eventually escalates, leaving the task unresolved.*

Model: Qwen2.5-3B + SEAL
Result: reward = 1.0

User Query: **My mobile data is unavailable or slow while I am abroad in France. I need excellent internet speed and have no Wi-Fi access.**

```
Assistant:
check_network_status

Tool/User Device:
Mobile Data Enabled: Yes
Data Roaming Enabled: No
Cellular Signal: excellent
Network Type: 5G

Assistant:
You are abroad, so data roaming should be enabled.

User/Device:
Data Roaming is now ON.
Status Bar: Excellent | 5G | Data Enabled | Data Saver

Assistant:
run_speed_test

Tool/User Device:
Speed Test Result: 55.00 Mbps (Good).
```

```
User:
Good is not enough. I need excellent speed.

Assistant:
The issue is not fully resolved. I will check whether data
    restrictions are
limiting your speed.

Assistant:
check_data_restriction_status

Tool/User Device:
Data Saver mode is ON.

Assistant:
Please turn off Data Saver mode.

User/Device:
Data Saver Mode is now OFF.

Assistant:
run_speed_test

Tool/User Device:
Speed Test Result: 275.00 Mbps (Excellent).

User:
The speed is now excellent. The issue is resolved.
###STOP###
```

Note (Success Point): *SEAL does not stop after the first partial fix. Once roaming improves the connection only to* Good*, the model continues diagnosis, checks data restrictions, identifies Data Saver as the remaining bottleneck, and reaches* Excellent *speed.*

H Supplementary Experimental Analyses

H.1 Single-Run Comparisons and Factorial Controls

Protocol. Tables 2 and 4 report one training run per condition. BFCL V3 evaluation contains 400 examples, 100 in each category. Category accuracy is the percentage of successful examples, and the overall score is the arithmetic mean of the four categories. A single successful example changes category accuracy by 1.00 point and overall accuracy by 0.25 points. All training-only additions are removed at evaluation.

Checkpoint selection. Checkpoint selection uses a separate development set that is disjoint from the 400-example BFCL V3 evaluation split. The reported evaluation set is used only for final evaluation and does not participate in checkpoint selection. The same selection protocol is applied across the matched training conditions.

Environment Tuning (Lu et al., 2026) is evaluated under the same backbone, data split, and evaluation protocol. It follows its released curriculum, actionable feedback, and progress-reward recipe as a method-level baseline. Vanilla RL supplies the matched binary-reward control.

Qwen2.5-7B results. The SEAL reference is 161/400 (40.25%), with Base, Missing Functions, Missing Parameters, and Long Context scores of 58.00, 36.00, 34.00, and 33.00%, respectively. Vanilla RL scores 123/400 (30.75%), and Environment Tuning scores 37.00%. The corresponding differences are 9.50 and 3.25 percentage points.

Full SEAL and factorial controls. Table 4 crosses two interface policies with two advantage-weighting policies. Static+Uniform fixes the enriched training interface and weights verifier failures by 1.5 and successes by 1.0, independently of diagnosis identity. Dynamic+Uniform enables online diagnosis-conditioned recovery selection while retaining that uniform weighting rule. Static+Diagnosis-aware fixes the interface while using the diagnostic utilities and updates of Section 3.4. Full SEAL, Dynamic+Diagnosis-aware, enables both online recovery selection and diagnosis-aware weighting. All four conditions share the base schema affordances, training split, GRPO backbone, rollout count, binary verifier reward, and development-set checkpoint-selection protocol. Vanilla RL is an external baseline, rather than the Static+Uniform cell.

The four BFCL category macro-averages are 34.75%, 34.50%, 35.50%, and 40.25% for Static+Uniform, Dynamic+Uniform, Static+Diagnosis-aware, and Full SEAL, respectively. Full SEAL's gains over the three controls are 5.50, 5.75, and 4.75 percentage points. The interaction point estimate on the accuracy scale is

$$I = 40.25 - 35.50 - 34.50 + 34.75 = +5.00 \text{ percentage points.}$$

These contrasts are run-level point estimates. Characterizing the run-to-run variability of the interaction remains a direction for larger-scale evaluation.

H.2 Blinded Diagnosis Audit

Auxiliary audit protocol. The diagnosis audit and failure counts use a separate seed-42 training run with two-stage, within-iteration rollout collection and bounded $+0.25$ credit on verifier-accepted focused turns. These auxiliary observations assess the shared diagnostic rules and operational failure patterns; the main Full-online configuration is specified in Appendix E.5.

We analyzed a label-stratified sample of 60 independently readable diagnoses from a corrected 7B training run: ten instances for each of the six observed outcomes. No `instance_mismatch` occurred. The first-pass annotation was performed by an AI assistant blinded to SEAL's assigned label, with access to the official dialogue, calls, initial state, cumulative prefix, and execution evidence. We therefore report an *AI-assisted blinded agreement audit*. A rendering error that joined a schema-reveal subturn to an earlier abstention turn was corrected before final scoring.

Table 6: First-pass exact-label agreement in the AI-assisted blinded audit of the auxiliary training run. Wilson intervals describe the sampled cases; equal sampling by label does not estimate the deployment-weighted diagnosis accuracy.

Outcome	Agreement	Rate (%)	Wilson 95% interval
`correct_abstention`	10/10	100.0	[72.2, 100.0]
`empty_turn_model_response`	10/10	100.0	[72.2, 100.0]
`pass`	10/10	100.0	[72.2, 100.0]
`response_mismatch`	7/10	70.0	[39.7, 89.2]
`spurious_tool_call`	10/10	100.0	[72.2, 100.0]
`state_mismatch`	10/10	100.0	[72.2, 100.0]
Total	57/60	95.0	[86.3, 98.3]

Failure-versus-nonfailure agreement is 59/60 (98.3%; Wilson interval [91.1%, 99.7%]). The three exact-label disagreements concern the difference between matching final state and matching required execution responses. For example, omitting an idempotent watchlist addition can preserve state but omit its required response; booking then cancelling and never booking can both leave an empty record but yield different responses. Subsequent inspection of BFCL semantics supported the checker labels in these cases. We retain the first-pass 57/60 figure rather than converting that reconciliation into an independent 100% accuracy claim. The small, label-stratified sample limits what this audit establishes about population-wide diagnosis quality.

H.3 Failure-Category Evolution

For this auxiliary training run, we summarize all 18,432 on-policy trajectories in six non-overlapping 12-step windows (3,072 trajectories each). Failure counts include the five model-error labels; `pass`,

`correct_abstention`, and bookkeeping errors are excluded. Each count is a turn-label occurrence, not a count of independent causal failures or held-out evaluation errors.

Table 7: Training-time dynamics of the auxiliary seed-42 run. Verifier success is trajectory-level; turn pass is computed over scorable turns. Failure traj. is the percentage containing at least one model-error label; it is a diagnostic statistic, not the complement of verifier success. No `instance_mismatch` occurs in these windows.

Steps	Success (%)	Turn pass (%)	Failure traj. (%)	Failures	Spurious	Empty	State	Response
1–12	25.1	45.9	82.7	7,301	1,229	1,240	4,148	684
13–24	34.6	55.7	75.2	6,203	1,220	1,181	3,225	577
25–36	39.4	59.5	71.9	5,797	1,253	1,125	2,992	427
37–48	46.0	63.9	67.8	5,366	1,289	1,033	2,692	352
49–60	50.9	68.0	65.0	4,821	1,236	770	2,493	322
61–72	54.1	69.5	64.0	4,667	1,254	866	2,296	251

From the first to the last window, total failure occurrences decrease 36.1%, state mismatches 44.6%, response mismatches 63.3%, and empty responses 30.2%. Spurious calls remain approximately flat (1,229 to 1,254); their share increases because the other categories contract. Verifier success rises from 25.1% to 54.1%. These are on-policy training observations and do not, by themselves, measure generalization or isolate the effect of online interface selection.

H.4 Runtime and Token Overhead

Table 8 summarizes timing over 61 ordinary, non-validation optimizer steps per condition. Median step time is 301.6 s for the online-interface configuration, compared with 281.7 s for Vanilla RL and 290.5 s for the static enriched interface: increases of 19.9 s (7.1%) and 11.1 s (3.8%), respectively. Rollout generation and policy optimization dominate the overall runtime. These descriptive timing summaries do not isolate the cost of individual components or establish identical context limits and rollout-length distributions across runs.

Table 8: Ordinary-step timing summary, excluding validation. The measurements summarize separate timing runs for the three listed configurations; auxiliary token counts and CPU measurements are reported below.

Condition	Median ordinary step (s)
Vanilla RL	281.7
Static enriched interface	290.5
Online-interface configuration	301.6

Separately, archived token measurements give 4,947.0 mean initial prompt tokens for Vanilla and 5,247.9 for Static: approximately 300.9 more tokens (6.08%). The auxiliary audit run has the same initial prompt length as Static; dynamic recovery guidance appears in 843/18,432 trajectories (4.57%) and contributes 187,730 tokens in total: 10.19 per trajectory averaged over all rollouts, or 222.7 per cued trajectory. This is 0.194% of the mean enriched initial prompt length, not a percentage of total training compute or total input tokens.

Archived reward-plus-advantage processing takes 16 ms median for Static and 35 ms for the auxiliary audit run. The 19 ms difference is approximately 0.0063% of a 301.6 s step-time scale; this arithmetic ratio combines separate summaries and is not an isolated causal-overhead estimate. A CPU microbenchmark of 256 capability observations, one utility update, and one atomic state save takes 4.69 ms median and 5.08 ms at the 95th percentile over 200 measured repetitions. This microbenchmark excludes rollout inference and tool execution. All training-only guidance and interface-selection logic are removed during evaluation.