THE ENVIRONMENT KNOWS WHY: BIDIRECTIONAL SELF-DISTILLATION FROM AUGMENTED TOOL FEEDBACK FOR AGENTIC RL

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-distillation with privileged context has emerged as a promising paradigm for agentic reinforcement learning, providing denser supervision for long-horizon credit assignment. However, existing methods typically construct such context for individual tasks or trajectories, limiting reuse and requiring repeated generation during training. We observe that rejected tool calls expose a reusable source of local supervision: the relevant tool contract is encoded in the environment implementation but often omitted from the returned feedback. Building on this observation, we introduce BEAD (*Bidirectional Environment-Augmented Distillation*), which combines task-agnostic environment augmentation with bidirectional feedback self-distillation. The former mines tool contracts offline and converts rejected feedback into reusable local guidance across tasks and trajectories. The latter uses the same augmented feedback bidirectionally: backward scoring provides hindsight correction for the rejected call, while forward scoring provides recovery guidance for the subsequent action. The resulting token-level teacher–student log-probability gaps serve as self-distillation signals for RL advantage reweighting. Extensive experiments demonstrate the effectiveness of BEAD across both in-distribution and out-of-distribution multi-turn tool-use settings, while analyses support the complementary roles of backward and forward supervision.

1 INTRODUCTION

Reinforcement learning with verifiable rewards has become a common approach for post-training large language model (LLM) agents that interact with tools (Shao et al., 2024; Guo et al., 2025; Yu et al., 2025; Qian et al., 2025). In multi-turn settings, however, an agent may issue many interdependent tool calls before receiving an outcome reward (Yao et al., 2023; Patil et al., 2025; Barres et al., 2025), making credit assignment difficult (Zhang, 2026). The same reward is propagated across decisions that contribute differently to the final outcome. When a tool call is rejected, this signal neither identifies the problematic decision nor indicates how the agent should recover.

Recently, several works have addressed this problem through on-policy self-distillation with privileged context (Zhao et al., 2026; Hübotter et al., 2026; Wang et al., 2026a; Lu et al., 2026b). A teacher scores student-generated actions while conditioning on information unavailable to the acting policy, such as successful trajectories, reference answers, learned skills, or hindsight from later in the episode (Wang et al., 2026a; Lu et al., 2026c; Li et al., 2026). This provides more localized supervision than a single outcome reward, but the privileged context is typically constructed from the task or trajectory being trained on. Task-level context remains coarse for individual decisions (Wang et al., 2026b; Lu et al., 2026c), whereas turn-level feedback is more precise but often requires trajectory-specific generation (Yang et al., 2026b; Yeo et al., 2026; Jin et al., 2026). As illustrated in Figure 1, more localized supervision therefore comes at the cost of repeated task- or trajectory-specific construction. This raises a natural question: **Can turn-level privileged supervision be both local and reusable, without being reconstructed for every task or trajectory?**

To obtain such supervision, we turn to the environment itself. Rejected tool calls provide a natural entry point: the environment has already determined that the issued call is invalid. Specifically, a rejection arises after the environment evaluates the call against internal validity conditions, such as

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.

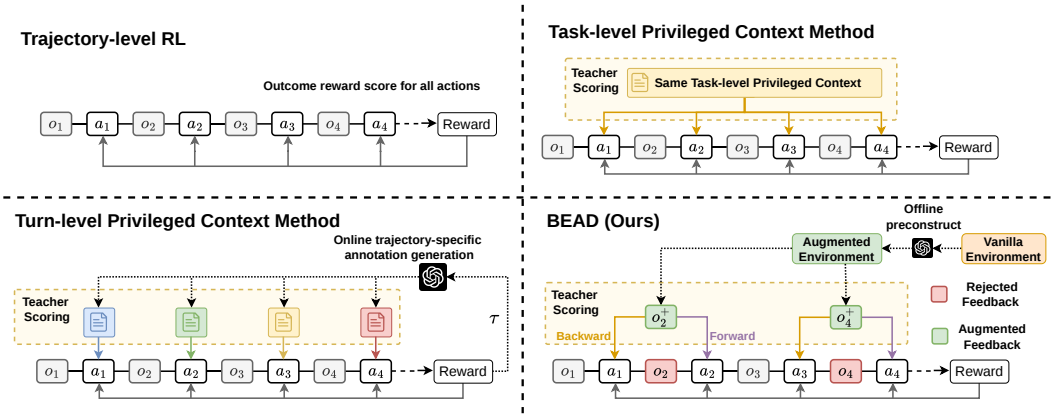


Figure 1: **Sources and granularity of supervision in multi-turn agent RL.** Existing approaches use trajectory-level rewards or task- and turn-level privileged context. BEAD instead preconstructs task-agnostic feedback from the environment and reuses each augmented rejection bidirectionally, *backward* for the rejected call and *forward* for the subsequent recovery.

argument schemas, admissible values, cross-tool dependencies, or state preconditions. Although the returned feedback may reveal little beyond the rejection, the surrounding environment implementation contains information about why the call is invalid and how the tool is intended to be used (Krishnamurthi & Flatt, 2026; Lu et al., 2026a). Because these constraints govern tool usage rather than any particular task solution, the relevant tool contract can be mined offline and reused whenever the same rejection pattern appears across tasks and trajectories.

Motivated by this, we introduce BEAD (*Bidirectional Environment-Augmented Distillation*), which turns environment knowledge into reusable local supervision for agentic RL. BEAD first performs *task-agnostic environment augmentation*, mining tool contracts offline to construct informative feedback that can be reused across tasks and trajectories. It then performs *bidirectional feedback self-distillation*, where the augmented feedback serves as privileged context for teacher scoring in two directions: backward scoring provides hindsight correction for the rejected call, while forward scoring evaluates the subsequent action to guide recovery. The resulting bidirectional teacher signals reweight the RL advantage, providing localized supervision for both correcting rejected calls and guiding subsequent recovery.

Empirically, BEAD consistently improves multi-turn tool use across in-distribution and out-of-distribution settings, including unseen agentic environments. Further analyses show reuse of offline augmentations across tasks and trajectories and support the complementary roles of backward correction and forward recovery guidance. Our contributions are as follows:

- **We identify** the environment as a reusable source of local supervision for rejected tool calls: tool contracts can be mined once and reused across tasks and trajectories.
- **We propose** BEAD, combining task-agnostic environment augmentation with bidirectional feedback self-distillation. Environment-derived augmentations provide hindsight correction for rejected calls and guide subsequent recovery, enabling localized supervision around the rejection.
- **We demonstrate** the effectiveness of BEAD on BFCL v4, τ^2-bench, and ACEBench with Qwen3-4B and Qwen3-8B. BEAD consistently outperforms GRPO with user-turn progress reward and privileged-context baselines across in-distribution and out-of-distribution settings.

2 Related Work

Reinforcement learning for multi-turn tool agents. Reinforcement learning with verifiable rewards is a common approach for post-training tool-using language agents (Shao et al., 2024; Yu et al., 2025; Qian et al., 2025). In multi-turn settings, however, agents interact with stateful tools over long horizons while receiving only task-level or terminal supervision (Patil et al., 2025; Barres et al., 2025; Wan et al., 2026), leaving credit assignment difficult (Zhang, 2026; Tao et al., 2026).

Recent methods address this with turn-level rewards or finer-grained advantage estimation over sub-trajectories (Feng et al., 2025; Cheng et al., 2026; Yao et al., 2026). These approaches refine how credit is estimated from reward, but remain limited by the information contained in the reward itself.

Self-distillation with privileged context. A complementary line addresses the same credit-assignment problem by introducing an additional information source. On-policy self-distillation conditions a self-teacher on privileged context unavailable to the acting policy, providing denser supervision over student-generated trajectories (Zhao et al., 2026; Hübotter et al., 2026; Da et al., 2025). For agents, such context can take the form of reference answers, successful trajectories, skills, or procedural memory (Wang et al., 2026a; Lu et al., 2026c; Xia et al., 2026; Yang et al., 2026a; Lu et al., 2026b), as well as hindsight or turn-level feedback targeted to individual decisions (Yeo et al., 2026). These signals provide more localized supervision than trajectory-level rewards, but are typically tied to the task, trajectory, or decision from which they are constructed, requiring repeated generation or selection during training.

Environment feedback augmentation. The environment provides a related source of local guidance without relying on a reference solution or successful trajectory. Prior work enriches environment feedback or organizes environment-side knowledge to provide more actionable guidance (Krishnamurthi & Flatt, 2026; Song et al., 2026; Zhang et al., 2026; Panthi & Abdelfattah, 2026; Reddy et al., 2026; Cui et al., 2025; Huang et al., 2026; Tian et al., 2026). Feedback-enriched approaches expose such guidance directly to the acting policy during training (Lu et al., 2026a; Yuan et al., 2026; Choudhury & Sodhi, 2025), creating a train–inference context mismatch when the augmented feedback is unavailable at deployment. BEAD instead uses augmented feedback as privileged teacher context, while the student always interacts with the original environment.

3 PRELIMINARIES

Problem formulation. We model each multi-turn tool-use task as a POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{E}, R)$. A task instance $x = (\Sigma_{\text{tool}}, o_1^{\text{usr}}, \ldots, o_N^{\text{usr}}) \sim \mathcal{D}$ consists of a tool set and N user requests revealed sequentially. At turn i, the agent observes o_i and generates a response $a_i = (y_{i,1}, \ldots, y_{i,L_i})$, where $y_{i,t} \sim \pi_\theta(\cdot \mid h_i, y_{i,<t})$ and $h_i = (o_1, a_1, \ldots, a_{i-1}, o_i)$. An observation is either a user request o^{usr} or an environment return $o_i^{\mathcal{E}} = \mathcal{E}(a_{i-1}, h_{i-1})$ following a tool call in a_{i-1}. When a tool call is rejected by the environment, we refer to it as a *rejected call* and to the corresponding environment return as *rejection feedback*. The agent interacts with the environment until a response contains no tool call, completing the current request and revealing the next. A trajectory therefore takes the form

$$o_1^{\text{usr}} \to a_1 \to o_2^{\mathcal{E}} \to a_2 \to \cdots \to o_k^{\mathcal{E}} \to a_k \to o_{k+1}^{\text{usr}} \to a_{k+1} \to \cdots \to a_n, \tag{1}$$

where a_k completes the first request and o_{k+1}^{usr} begins the next. For long-horizon agent training, we adopt the user-turn progress reward of Lu et al. (2026a), $R_{\text{P}} = \frac{1}{N}\sum_{j=1}^{N} r_j^{\text{state}} r_j^{\text{exec}}$, where r_j^{state} and r_j^{exec} evaluate the environment state and execution result for the j-th user request. The objective is $J(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, \tau\sim\pi_\theta}\left[R_{\text{P}}(\tau)\right]$.

Policy optimization. Following GRPO (Shao et al., 2024), we sample G rollouts for each task and normalize their progress rewards within the group to obtain the episode-level advantage

$$\hat{A}_{i,t} = \hat{A}^{\text{ep}} = \frac{R_{\text{P}}(\tau) - \mu}{\sigma}, \tag{2}$$

where μ and σ are the group mean and standard deviation. We optimize π_θ with the clipped surrogate

$$\mathcal{L}_{\text{RL}}(\theta) = -\mathbb{E}_{\tau,i,t}\Big[\min\Big(\varrho_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}\left(\varrho_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\Big)\Big], \tag{3}$$

where $\varrho_{i,t}(\theta) = \pi_\theta(y_{i,t} \mid h_i, y_{i,<t}) / \text{sg}(\pi_{\theta_{\text{old}}}(y_{i,t} \mid h_i, y_{i,<t}))$ is the token-level importance ratio.

4 BIDIRECTIONAL ENVIRONMENT-AUGMENTED DISTILLATION

BEAD turns environment rejections into dense supervision for multi-turn tool-use RL. Task-agnostic environment augmentation (§4.2) mines tool contracts behind rejected calls, producing feedback

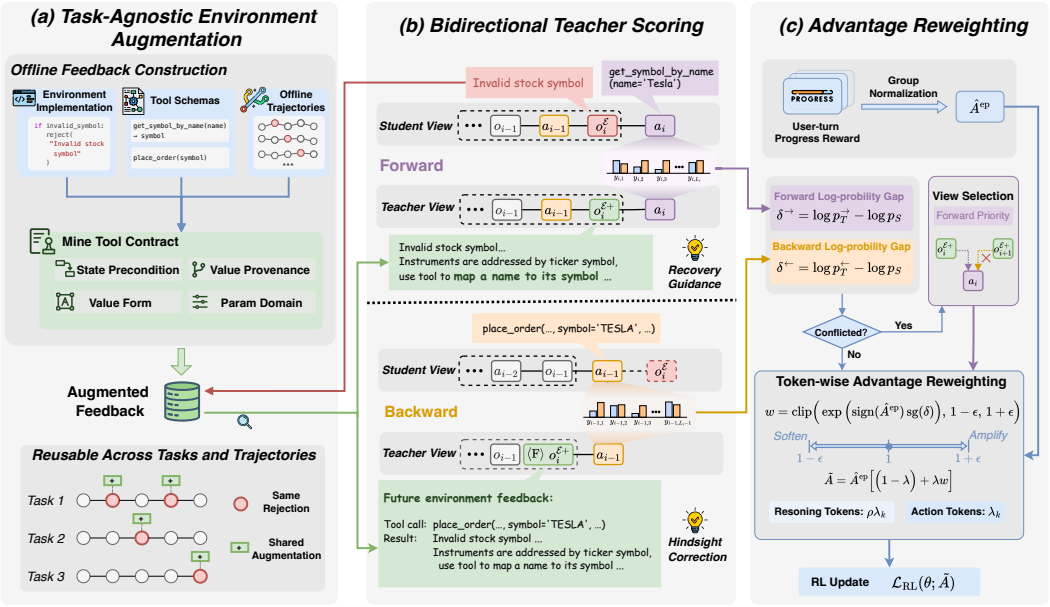


Figure 2: **BEAD.** *(a)* We construct task-agnostic augmentations offline from tool contracts encoded in the environment and reuse them across tasks and trajectories. *(b)* Augmented feedback enables bidirectional teacher scoring: backward for hindsight correction of rejected calls and forward for recovery guidance of subsequent actions. *(c)* Token-level teacher–student log-probability gaps provide the self-distillation signal for RL advantage reweighting.

reusable across tasks and trajectories. Conditioned on this privileged feedback, the teacher scores actions bidirectionally: backward for the rejected call and forward for subsequent recovery (§4.2). The resulting token-level log-probability gaps reweight the RL advantage (§4.3), focusing updates on tokens most informed by the augmentation. Figure 2 summarizes the full pipeline.

4.1 TASK-AGNOSTIC ENVIRONMENT AUGMENTATION

A rejection reveals that a tool call violates the environment's internal validity conditions, but the returned feedback may not expose the underlying cause. We therefore use rejection feedback as an entry point to mine the relevant tool contract from the surrounding environment implementation, tool schemas, and dependencies, and expose it as reusable local guidance:

$$o_i^{\mathcal{E}+} = \mathrm{Aug}\big(o_i^{\mathcal{E}}\big), \tag{4}$$

If no useful contract can be identified, the feedback is left unchanged. For example, "`Invalid stock symbol`" only reports the rejection, whereas the augmentation can reveal that the tool expects a ticker symbol and that a company name must first be mapped to its corresponding symbol.

We construct the augmentations once offline from the environment. Rejections sharing the same signature are grouped to share a single augmentation. For each signature, the source code surrounding the rejection, relevant tool schemas, and cross-tool or state dependencies are used to mine the underlying tool contract and construct the corresponding feedback. These contracts mainly capture parameter domains (e.g., an argument must fall within a fixed set or range), value formats (e.g., a filename rather than a full path), value provenance (e.g., a system-issued identifier), and state preconditions (e.g., a required state must be established by an earlier call). We retain only feedback that is grounded, informative, reachable, task-agnostic, and free of answer leakage. Because such feedback describes how a tool can be used rather than how a particular task should be solved, the same augmentation can be reused across tasks and trajectories and remains fixed throughout training. Full construction and validation details are provided in Appendix C.

4.2 BIDIRECTIONAL TEACHER SCORING

For each rejection feedback $o_i^{\mathcal{E}}$ that admits an augmentation $o_i^{\mathcal{E}+}$, the feedback lies between two actions: the rejected call in a_{i-1} and the subsequent action a_i. BEAD uses the same augmented feedback in two complementary directions. *Backward scoring* treats $o_i^{\mathcal{E}+}$ as hindsight when scoring a_{i-1}, measuring how the additional feedback changes the teacher's assessment of the rejected call. *Forward scoring* instead replaces $o_i^{\mathcal{E}}$ with $o_i^{\mathcal{E}+}$ when scoring a_i, measuring its effect on the subsequent action. Thus, the two directions use the same feedback but supervise different actions on opposite sides of the rejection.

In both directions, the teacher is the current policy with stop-gradient parameters, $\pi_{\bar{\theta}}$ with $\bar{\theta} = \mathrm{sg}(\theta)$, and scores student-sampled actions under privileged context unavailable to the acting policy. The corresponding contexts are

$$\begin{aligned} \text{forward: } \tilde{h}_i^{\rightarrow} &= \big(o_1, a_1, \ldots, a_{i-1}, o_i^{\mathcal{E}+}\big), \qquad h_i = \big(o_1, a_1, \ldots, a_{i-1}, o_i^{\mathcal{E}}\big), \text{ scored: } a_i, \\ \text{backward: } \tilde{h}_{i-1}^{\leftarrow} &= \big(o_1, a_1, \ldots, o_{i-1}, \langle F \rangle o_i^{\mathcal{E}+}\big), \; h_{i-1} = \big(o_1, a_1, \ldots, o_{i-1}\big), \text{ scored: } a_{i-1}. \end{aligned} \tag{5}$$

where $\langle F \rangle$ is the fixed header "Future environment feedback:". The corresponding token-level log-probability gaps are

$$\begin{aligned} \delta_{i,t}^{\rightarrow} &= \log \pi_{\bar{\theta}}\big(y_{i,t} \mid \tilde{h}_i^{\rightarrow},\, y_{i,<t}\big) - \log \pi_{\bar{\theta}}\big(y_{i,t} \mid h_i,\, y_{i,<t}\big), \\ \delta_{i-1,t}^{\leftarrow} &= \log \pi_{\bar{\theta}}\big(y_{i-1,t} \mid \tilde{h}_{i-1}^{\leftarrow},\, y_{i-1,<t}\big) - \log \pi_{\bar{\theta}}\big(y_{i-1,t} \mid h_{i-1},\, y_{i-1,<t}\big). \end{aligned} \tag{6}$$

Within each direction, the two terms score the same student-sampled token under the same stop-gradient policy, but with the teacher and student contexts defined above. Accordingly, $\delta_{i-1,t}^{\leftarrow}$ measures the hindsight effect on the rejected action, whereas $\delta_{i,t}^{\rightarrow}$ measures the guidance effect on the subsequent action. Positive and negative gaps indicate increased and decreased teacher support for the corresponding token, respectively. We cap each trajectory at B distinct augmented feedback items to bound additional teacher computation.

Direction selection. Let $\mathcal{I}_{\mathrm{aug}}$ denote the positions at which an augmentation is triggered. An action may be eligible for both directions: a_i can be scored forward using the preceding augmented feedback $o_i^{\mathcal{E}+}$, while also being scored backward if its own tool call produces another augmented rejection $o_{i+1}^{\mathcal{E}+}$. Applying both gaps to the same action would compound their effects, so BEAD keeps at most one direction:

$$d(i) = \begin{cases} \rightarrow, & i \in \mathcal{I}_{\mathrm{aug}}, \\ \leftarrow, & i \notin \mathcal{I}_{\mathrm{aug}} \text{ and } i+1 \in \mathcal{I}_{\mathrm{aug}}, \\ \varnothing, & \text{otherwise}, \end{cases} \tag{7}$$

We let $\mathcal{M}$ denote response tokens of actions with $d(i) \neq \varnothing$. We prioritize forward scoring when both directions are available. Here, a_i is the policy's response to the preceding rejection, so the forward signal directly evaluates the action against the recovery guidance associated with that rejection. Forward priority therefore preserves this recovery-sensitive supervision on the action immediately following a rejection, while backward scoring remains available for rejected calls not selected in the forward direction. This choice is consistent with Fig. 4(b,c): backward scoring is predominantly corrective, whereas forward scoring separates successful recovery from continued rejection.

4.3 ADVANTAGE REWEIGHTING

Let $\hat{A}_{i,t}$ be the whitened group-relative advantage exactly as the RL pipeline leaves it, so that $\mathrm{sign}(\hat{A}_{i,t})$ is the direction the reward asks the update to move in. Following Yang et al. (2026a), we convert the token-level gap $\delta_{i,t}$ into a direction-aware weight that reweights the group-normalized RL advantage $\hat{A}_{i,t}$:

$$w_{i,t} = \mathrm{clip}\Big(\exp\big(\mathrm{sign}(\hat{A}_{i,t})\ \mathrm{sg}(\delta_{i,t})\big),\, 1-\varepsilon_w,\, 1+\varepsilon_w\Big), \quad \tilde{A}_{i,t} = \hat{A}_{i,t}\Big[\big(1-\lambda_{i,t}\big)+\lambda_{i,t}\, w_{i,t}\Big]. \tag{8}$$

The sign of the RL advantage determines how the teacher evidence is used. When $\hat{A}_{i,t} > 0$, tokens supported by the augmented feedback receive larger positive updates, while disfavored tokens are

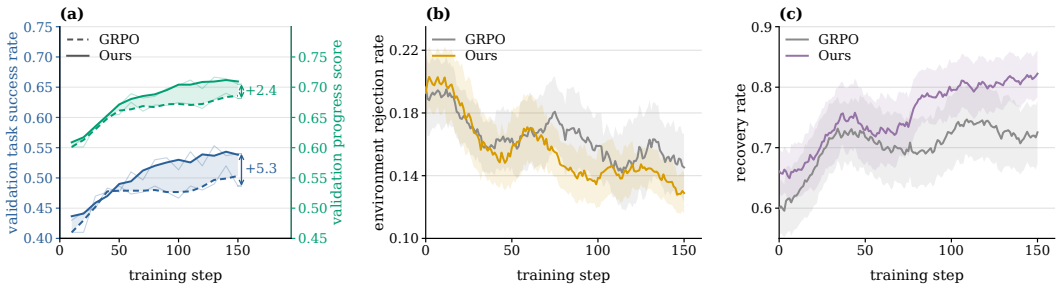


Figure 3: Training dynamics of BEAD compared with GRPO on Qwen3-4B. **(a)** Validation task success rate and user-turn progress score. **(b)** Environment rejection rate, measuring how often tool calls are rejected during interaction. **(c)** Recovery rate, measuring how often the subsequent action successfully resolves a rejection.

attenuated. When $\hat{A}_{i,t} < 0$, the effect is reversed: disfavored tokens receive stronger penalties, while supported tokens are penalized less. Since $w_{i,t} > 0$, reweighting never changes the sign of the original advantage; the environment reward determines the direction of the update, while the teacher redistributes its magnitude across tokens.

We modulate the reweighting strength by token type,

$$\lambda_{i,t} = \lambda_k \, s_{i,t}, \qquad s_{i,t} = \begin{cases} 1 & y_{i,t} \text{ is an action token,} \\ \rho & y_{i,t} \text{ is a reasoning token,} \end{cases} \qquad (i,t) \in \mathcal{M}, \tag{9}$$

We downweight reasoning tokens because augmented feedback is most directly tied to executable tool actions, while longer reasoning spans can otherwise receive disproportionate reweighting despite weaker attribution. Clipping limits the influence of extreme teacher–student gaps, while $\lambda_k = \lambda_0(1 - k/K)_+$ gradually anneals the teacher contribution during training. The resulting $\tilde{A}_{i,t}$ directly replaces $\hat{A}_{i,t}$ in the RL objective equation 3; augmented feedback is used only for teacher scoring and is not required at inference time.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate on the multi-turn setting of BFCL v4 (Patil et al., 2025). `Base`, `Missing Functions`, and `Missing Parameters` form the in-distribution setting, with half of each subset randomly sampled for training and the remainder reserved for held-out evaluation; the augmentation table is constructed solely from the training split. For out-of-distribution evaluation, we use `Multi-turn Long Context`, `Web Search`, and `Memory` from BFCL v4, together with τ^2-bench (Barres et al., 2025) and the ACEBench Agent split (Chen et al., 2025), covering held-out capabilities as well as unseen environments, tool interfaces, and task formats. Results are averaged over three independent evaluation runs using each benchmark's official metrics. Further details are provided in Appendix B.

Baselines. We compare BEAD with GRPO (Shao et al., 2024) and representative methods that exploit privileged context during training. EnvTuning (Lu et al., 2026a) uses actionable environment augmentation as an intermediate stage in a multi-stage curriculum, while Skill0 (Lu et al., 2026c) progressively removes privileged context to internalize it. Skill-SD (Wang et al., 2026a), RLSD (Yang et al., 2026a), and SDAR (Lu et al., 2026b) incorporate self-distillation with privileged teacher context. All methods share the same backbone, training split, optimization-step budget, and training framework. Implementation details are provided in Appendix D.3.

Implementation Details. We use Qwen3-4B and Qwen3-8B (Yang et al., 2025) with Thinking Mode enabled throughout training and evaluation. All methods are implemented with ROLL (Wang et al., 2025) and trained for 150 optimization steps. For BEAD, we allow at most $B = 8$ distinct augmented feedback items per trajectory and prioritize forward scoring when both directions apply.

Table 1: **Main results on BFCL v4.** We report performance on the in-distribution (ID) multi-turn setting and the out-of-distribution (OOD) tracks. parenthesized values denote absolute improvements over the corresponding Vanilla backbone. **Bold** and underlined values indicate the best and second-best results in each column, respectively.

Methods	BFCL v4 (ID)				BFCL v4 (OOD)			
	Base	Miss Func	Miss Param	Avg.	Long Context	Web Search	Memory	Avg.
Qwen3-4B(Thinking)								
Vanilla	37.33	33.00	30.67	33.67	33.50	7.50	16.49	19.16
GRPO	53.00	43.67	36.33	44.33 (↑+10.66)	44.83	11.67	20.72	25.74 (↑+6.58)
EnvTuning	55.00	48.00	43.33	48.78 (↑+15.11)	44.67	8.83	21.58	25.03 (↑+5.87)
Skill0	53.33	49.00	42.33	48.22 (↑+14.55)	42.83	10.33	19.43	24.20 (↑+5.04)
Skill-SD	55.00	47.33	43.67	48.67 (↑+15.00)	44.17	15.50	19.50	26.39 (↑+7.23)
RLSD	55.33	45.00	40.00	46.78 (↑+13.11)	42.50	13.83	19.07	25.13 (↑+5.97)
SDAR	52.00	45.00	39.00	45.33 (↑+11.66)	43.00	10.67	18.78	24.15 (↑+4.99)
BEAD (Ours)	**57.33**	**52.00**	**45.33**	**51.56** (↑+17.89)	**46.17**	**16.00**	**21.65**	**27.94** (↑+8.78)
Qwen3-8B(Thinking)								
Vanilla	48.33	41.00	39.33	42.89	42.00	12.50	19.35	24.62
GRPO	57.00	55.33	44.00	52.11 (↑+9.22)	51.17	12.17	20.36	27.90 (↑+3.28)
EnvTuning	63.33	57.33	43.33	54.67 (↑+11.78)	52.83	13.67	19.50	28.67 (↑+4.05)
Skill0	59.67	56.00	46.33	54.00 (↑+11.11)	53.50	14.17	20.14	29.27 (↑+4.65)
Skill-SD	59.00	54.33	44.67	52.67 (↑+9.78)	52.67	14.00	19.50	28.72 (↑+4.10)
RLSD	59.00	51.67	44.33	51.67 (↑+8.78)	50.00	18.50	21.22	29.91 (↑+5.29)
SDAR	61.67	57.00	45.67	54.78 (↑+11.89)	52.67	14.00	21.00	29.22 (↑+4.60)
BEAD (Ours)	**64.00**	**58.00**	**47.67**	**56.56** (↑+13.67)	**54.00**	**20.17**	**21.94**	**32.04** (↑+7.42)

We set $\rho = 0.5$ and $\varepsilon_w = 0.5$, and linearly anneal the reweighting strength from $\lambda_0 = 0.5$ to zero throughout training. Remaining details are provided in Appendix D.

5.2 Overall Performance

BEAD consistently improves multi-turn tool use within and beyond the training distribution. As shown in Table 1, BEAD achieves the highest average performance on the BFCL v4 in-distribution setting at both model scales, reaching 51.56% with Qwen3-4B and 56.56% with Qwen3-8B. Relative to the corresponding Vanilla backbones, these results represent absolute improvements of 17.89% and 13.67%, respectively. The gains are consistent across `Base`, `Missing Functions`, and `Missing Parameters`, rather than being concentrated in a particular subset. BEAD also maintains its advantage on the held-out BFCL tracks that are never used for training, achieving OOD averages of 27.94% and 32.04%, corresponding to improvements of 8.78% and 7.42% over the respective Vanilla models. Its consistent gains across `Long Context`, `Web Search`, and `Memory` indicate that the benefit extends beyond the capabilities directly observed during training.

The learned policy further transfers to entirely unseen agentic environments. Table 2 evaluates models trained only on BFCL v4 directly on τ^2-bench and ACEBench-Agent, which introduce unseen environments, tool interfaces, and task formats. BEAD achieves the highest overall performance on both benchmarks at both model scales. On τ^2-bench, it improves over the Vanilla backbone by 4.91% with Qwen3-4B and 6.02% with Qwen3-8B. On ACEBench-Agent, the corresponding improvements are even larger, reaching 9.34% and 9.00%. These results show that the benefits learned from environment-augmented supervision are not confined to the BFCL environment, but transfer to agentic settings with different tools and interaction structures.

BEAD exhibits systematic improvements in interaction behavior during training. As shown in Figure 3, compared with GRPO, BEAD maintains higher validation task success and user-turn progress scores, with final gains of 5.3% and 2.4%, respectively. Meanwhile, its environment rejection rate decreases more substantially, while the recovery rate following a rejection remains consistently higher. These trends are consistent with the intended role of BEAD: the policy becomes less

Table 2: **Out-of-distribution transfer to unseen agentic benchmarks.** All models are trained only on BFCL v4 and evaluated on τ^2-bench and ACEBench-Agent. **Bold** and underlined values indicate the best and second-best results in each column, respectively.

Methods	τ^2-bench				ACEBench-Agent		
	Retail	Airline	Telecom	Overall	Multi-Step	Multi-Turn	Overall
Qwen3-4B(Thinking)							
Vanilla	37.72	28.67	16.08	27.22	36.67	34.44	35.33
GRPO	39.47	36.00	15.50	29.02 (↑+1.80)	45.00	32.78	37.67 (↑+2.34)
EnvTuning	42.40	33.33	14.62	29.38 (↑+2.16)	44.17	38.33	40.67 (↑+5.34)
Skill0	40.94	38.67	14.33	29.62 (↑+2.40)	45.00	31.11	36.67 (↑+1.34)
Skill-SD	40.94	34.67	14.91	29.14 (↑+1.92)	45.83	38.89	41.67 (↑+6.34)
RLSD	**43.86**	32.00	12.87	29.02 (↑+1.80)	46.00	36.67	40.40 (↑+5.07)
SDAR	40.35	38.00	13.16	28.78 (↑+1.56)	48.33	29.44	37.00 (↑+1.67)
BEAD (Ours)	**43.86**	**39.33**	**17.25**	**32.13** (↑+4.91)	**52.50**	**39.44**	**44.67** (↑+9.34)
Qwen3-8B(Thinking)							
Vanilla	41.52	32.67	14.33	28.78	45.00	38.33	41.00
GRPO	48.83	35.33	16.96	33.33 (↑+4.55)	45.83	43.89	44.67 (↑+3.67)
EnvTuning	41.52	38.67	15.79	30.46 (↑+1.68)	47.50	**47.22**	47.33 (↑+6.33)
Skill0	42.11	35.33	14.62	29.62 (↑+0.84)	54.17	42.22	47.00 (↑+6.00)
Skill-SD	45.03	32.67	16.96	31.29 (↑+2.51)	53.33	41.67	46.33 (↑+5.33)
RLSD	**49.12**	29.33	14.91	31.53 (↑+2.75)	52.50	43.89	47.33 (↑+6.33)
SDAR	48.83	37.33	15.50	33.09 (↑+4.31)	50.83	41.67	45.33 (↑+4.33)
BEAD (Ours)	47.37	**40.00**	**18.13**	**34.80** (↑+6.02)	**57.50**	45.00	**50.00** (↑+9.00)

Table 3: **Ablation study on the in-distribution BFCL v4 multi-turn setting with Qwen3-4B.** We ablate the training signal, bidirectional scoring directions, augmentation, and direction priority; Δ Avg. reports the change in average performance relative to full BEAD.

Methods	BFCL v4 multi-turn				
	Base	Miss Func	Miss Param	Avg.	Δ Avg.
BEAD (Ours)	57.33	52.00	45.33	51.56	–
Training signal					
w/o progress reward *(binary outcome)*	57.33	49.33	41.67	49.44	-2.12
w/o λ decay *(constant λ)*	57.00	51.00	43.33	50.44	-1.12
Scoring directions					
w/o backward *(forward only)*	56.67	49.00	43.67	49.78	-1.78
w/o forward *(backward only)*	56.33	47.67	43.00	49.00	-2.56
▶ w/o augmentation	52.67	46.33	39.67	46.22	-5.34
Direction priority					
w/o forward priority *(backward priority)*	56.33	52.00	40.67	49.67	-1.89

prone to rejected tool calls while improving its ability to recover when rejections occur. Together, these results provide complementary evidence beyond final task performance, showing systematic changes in the policy's interaction behavior throughout optimization.

5.3 Ablation Study

Training signal. Replacing the user-turn progress reward with a binary outcome reward leads to a 2.12% decrease, while still substantially outperforming GRPO (44.33%), as shown in Table 3. This shows that BEAD remains effective without fine-grained progress rewards, which provide an additional complementary training signal. Removing the decay of the reweighting strength λ causes a further 1.12% drop, suggesting that teacher supervision is most useful early in training before optimization gradually returns to environment rewards.

Scoring directions and augmentation. Removing either teacher view degrades performance, with forward-only and backward-only yielding drops of 1.78% and 2.56%, respectively. This supports

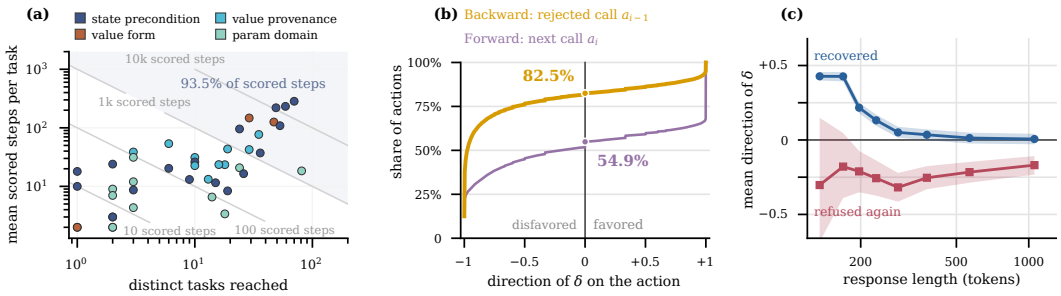


Figure 4: (a) Cross-task reuse of triggered augmentations, showing the number of distinct tasks reached and mean teacher-scored steps per task. (b) Empirical CDF of the action-level direction of token-wise δ under backward and forward teacher scoring. (c) Mean forward-view direction across response-length bins for recovered and refused-again actions; shaded regions denote bootstrap 95% confidence intervals.

the complementary roles of backward hindsight correction and forward recovery guidance. Starting from the backward-only variant, further removing augmentation incurs an additional 2.78% decrease, amounting to the largest overall drop (5.34%) in Table 3. This indicates that backward supervision benefits substantially from the tool-contract information exposed by environment augmentation, rather than from the original rejection alone.

Direction priority. Replacing forward priority with backward priority results in a 1.89% decrease. This supports prioritizing the forward view when both directions apply to the same action, preserving recovery-sensitive supervision while retaining backward scoring for other rejected calls.

5.4 Analysis of Reusable Bidirectional Supervision

Cross-task reuse. Figure 4(a) examines whether the offline augmentations are actually reused across training tasks. Among the 43 augmentations triggered during training, reuse is highly concentrated: only 11 account for 93.5% of all teacher-scored steps. Importantly, this concentration comes primarily from cross-task coverage rather than repeated use within individual tasks; these frequently triggered augmentations reach a median of 47 distinct tasks. This supports the central motivation of task-agnostic augmentation: a contract extracted once from the environment can provide supervision across many trajectories without trajectory-specific reconstruction.

Complementary Roles of Bidirectional Teacher Signals. Figure 4(b) shows that backward scoring is predominantly corrective: 82.5% of rejected calls are not favored by the augmented feedback, indicating that hindsight supervision tends to suppress the action responsible for the rejection. Figure 4(c) examines forward scoring. Because token-level δ is affected by response length, we group actions into eight equal-sized length bins and compare outcomes within each bin. Recovered actions receive a positive mean direction in all eight bins, whereas actions rejected again receive a negative mean direction in all eight, with a significant separation in every bin. Together, the two directions provide complementary signals for correcting rejected calls and guiding subsequent recovery.

6 Conclusion

We introduced BEAD, a framework for turning latent environment knowledge into reusable local supervision for multi-turn agent RL. Rather than reconstructing privileged context for individual tasks or trajectories, BEAD mines tool contracts from the environment offline and uses the resulting augmented feedback to score both the rejected call and the subsequent action. This bidirectional scoring turns the augmented feedback into localized supervision for policy learning. Across both in-distribution and out-of-distribution tool-use settings, BEAD consistently improves performance, while ablations and analyses confirm the importance of environment augmentation and the complementary roles of backward and forward supervision. More broadly, our results suggest that, beyond providing observations and rewards, the environment itself can serve as a reusable source of structured supervision for multi-turn agent learning.

AI USE STATEMENT

We used generative AI tools only to polish the writing of this paper.

ETHICS STATEMENT

This work uses only publicly available benchmarks and open-source models in simulated environments. It involves no human subjects, personal or sensitive data, or live external services, and complies with the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

The code and data used in this work are available in an anonymous repository to facilitate reproduction: `https://anonymous.4open.science/r/BEAD-B4BC`. Section 5.1 describes the experimental setup, including data splits, backbone models, training budgets, and key hyperparameters, while Appendix B details the benchmarks, reward design, and evaluation protocols. Appendix C documents the complete environment-augmentation pipeline, including rejection grouping, feedback generation and validation, replay checks, and the analyzer and validator prompts. Appendix D provides the full training configuration, system prompt, baseline reimplementations, and compute setup. All methods are implemented within the same training framework and use matched training splits and optimization budgets for controlled comparison.

REFERENCES

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.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Xinzhi Wang, and Wu Liu. ACEBench: A comprehensive evaluation of LLM tool usage. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 12970–12998, 2025. doi: 10.18653/v1/2025.findings-emnlp.697.

Xin Cheng, Shuo He, Lang Feng, HaiYang Xu, Ming Yan, Lei Feng, and Bo An. Beyond trajectory-level attribution: Graph-based credit assignment for agentic reinforcement learning. In *International Conference on Machine Learning*, 2026.

Sanjiban Choudhury and Paloma Sodhi. Better than your teacher: Llm agents that learn from privileged ai feedback. In *International Conference on Learning Representations*, 2025.

Yue Cui, Liuyi Yao, Shuchang Tao, Weijie Shi, Yaliang Li, Bolin Ding, and Xiaofang Zhou. Enhancing tool learning in large language models with hierarchical error checklists. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 16357–16375, 2025. doi: 10.18653/v1/2025.findings-acl.841.

Jeff Da, Clinton Wang, Xiang Deng, Yuntao Ma, Nikhil Barhate, and Sean Hendryx. Agent-RLVR: Training software engineering agents via guidance and environment rewards. *arXiv preprint arXiv:2506.11425*, 2025.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. In *Advances in Neural Information Processing Systems*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z. arXiv:2501.12948.

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

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Xiaolong Jin, Dingmin Wang, Vijay Lingam, and Varun Kumar. AHEAD: Adaptive hindsight with environment-augmented distillation for agentic rl. *arXiv preprint arXiv:2608.24114*, 2026.

Shriram Krishnamurthi and Matthew Flatt. Type-error ablation and ai coding agents. *arXiv preprint arXiv:2606.01522*, 2026.

Xiaozhe Li, Tianyi Lyu, Yang Li, Yichuan Ma, Peiji Li, Linyang Li, Qipeng Guo, Dahua Lin, and Kai Chen. What and when to distill: Selective hindsight distillation for multi-turn agents. *arXiv preprint arXiv:2605.19447*, 2026.

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 *International Conference on Learning Representations*, 2026a. arXiv:2510.10197.

Zhengxi Lu, Zhiyuan Yao, Zhuowen Han, Zi-Han Wang, Jinyang Wu, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Self-distilled agentic reinforcement learning. *arXiv preprint arXiv:2605.15155*, 2026b.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Skill0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026c.

Sugam Panthi and Rabab Abdelfattah. Outcome monitors: Recovery affordances for silent tool failures. *arXiv preprint arXiv:2608.19303*, 2026.

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 *International Conference on Machine Learning*, 2025.

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

Vikas Reddy, Sumanth Reddy Challaram, and Abhishek Basu. Reason less, verify more: Deterministic gates recover a silent policy-violation failure mode in tool-using llm agents. In *KDD Workshop on Evaluation and Trustworthiness of Agentic AI*, 2026. arXiv:2607.07405.

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

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. EnvScaler: Scaling tool-interactive environments for LLM agent via programmatic synthesis. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, 2026. doi: 10.18653/v1/2026.findings-acl.407.

Leitian Tao, Baolin Peng, Wenlin Yao, Tao Ge, Hao Cheng, Mike Hang Wang, Jianfeng Gao, and Sharon Li. Trace: Turn-level reward assignment via credit estimation for long-horizon agents. *arXiv preprint arXiv:2607.13988*, 2026.

Yang Tian, Zhengpeng Shi, Yu Zhou, and Bo Zhao. Beyond function calling: Benchmarking tool-using agents under tool-environment unreliability. *arXiv preprint arXiv:2606.25819*, 2026.

Zelin Wan, Arash Nourian, Xiaoxiao Li, Nihar Nandan, and Kamalakannan Nandagopal. Apiflow-bench: Measuring whether agents survive long, dependent API workflows. *arXiv preprint arXiv:2608.29128*, 2026.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, and Honggang Qi. Skill-SD: Skill-conditioned self-distillation for multi-turn LLM agents. *arXiv preprint arXiv:2604.10674*, 2026a.

Weixun Wang, Shaopan Xiong, Gengru Chen, Wei Gao, Sheng Guo, Yancheng He, Ju Huang, Jiaheng Liu, Zhendong Li, Xiaoyang Li, et al. Reinforcement learning optimization for large-scale learning: An efficient and user-friendly scaling library. *arXiv preprint arXiv:2506.06122*, 2025.

Zi-Han Wang, Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Jie Wu, Zhengzhou Cai, Yueqing Sun, Ziang Ye, Linji Hao, Qi Gu, Xunliang Cai, Yongliang Shen, and Yujiu Yang. Agentopsd: Recursive self-distillation for agentic reinforcement learning. *arXiv preprint arXiv:2608.05987*, 2026b.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled RLVR. *arXiv preprint arXiv:2604.03128*, 2026a.

Shuo Yang, Jinyang Wu, Zhengxi Lu, Yuhao Shen, Fan Zhang, Lang Feng, Shuai Zhang, Haoran Luo, Zheng Lian, Zhengqi Wen, and Jianhua Tao. Opid: On-policy skill distillation for agentic reinforcement learning. *arXiv preprint arXiv:2606.26790*, 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 *International Conference on Learning Representations*, 2023.

Xincheng Yao, Haobo Fu, Weiming Liu, and Chongyang Zhang. Back to the definition: Estimating step-level advantages via trajectory graphs for agentic reinforcement learning. *arXiv preprint arXiv:2609.28963*, 2026.

Woongyeong Yeo, Yumin Choi, Taekyung Ki, and Sung Ju Hwang. Hint-sd: Targeted hindsight self-distillation for long-horizon agents. *arXiv preprint arXiv:2605.17873*, 2026.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, 2025.

Hongbang Yuan, Zhuoran Jin, and Yixin Cao. Environments as scaffold: Enriching feedback to bootstrap self-evolving agents in long-horizon tasks. *arXiv preprint arXiv:2609.08404*, 2026.

Chenchen Zhang. From reasoning to agentic: Credit assignment in reinforcement learning for large language models. *arXiv preprint arXiv:2604.09459*, 2026.

Zeyu Zhang, Guohao Li, Zhenchang Xing, Alexandros Apostolopoulos, Yu Lin Lee, and Liang Zheng. Gecko: A simulation environment with stateful feedback for refining agent tool calls. In *International Conference on Machine Learning*, 2026.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Algorithm 1 BEAD

Require: policy π_θ, tasks $\mathcal{D}$, group size G, augmentation budget B, weight bound ε_w, reasoning scale ρ, teacher weight λ_0, steps K

$\mathcal{T} \leftarrow \textsc{MineContracts}(\mathcal{E})$ ▷ offline; App. C
for step $k = 1, \dots, K$ **do**
  $\lambda_k \leftarrow \lambda_0 \, (1 - k/K)_+$ ▷ anneal the teacher out
  ▷ Rollout in the unmodified environment
  $x \sim \mathcal{D}$; $\{\tau_i\}_{i=1}^{G} \sim \pi_\theta(\cdot \mid x)$ ▷ the student never sees $o^{\mathcal{E}+}$
  $\hat{A}_{i,t} \leftarrow \big(R_{\mathrm{P}}(\tau) - \mu\big)/\sigma$ ▷ Eq. 2
  ▷ Environment augmentation
  for each rejection observation $o_i^{\mathcal{E}}$ of τ **do**
    $o_i^{\mathcal{E}+} \leftarrow \mathrm{Aug}(o_i^{\mathcal{E}})$ by lookup in $\mathcal{T}$ ▷ Eq. 4
  end for
  $\mathcal{I}_{\mathrm{aug}} \leftarrow \{i : o_i^{\mathcal{E}}$ triggers an augmentation$\}$, one index per distinct statement, at most B
  ▷ Bidirectional teacher scoring
  for each action i **do**
    $d(i) \leftarrow (\rightarrow)$ if $i \in \mathcal{I}_{\mathrm{aug}}$; $(\leftarrow)$ if $i \notin \mathcal{I}_{\mathrm{aug}}$ and $i{+}1 \in \mathcal{I}_{\mathrm{aug}}$; else $\varnothing$ ▷ Eq. 7
    if $d(i) \neq \varnothing$ **then**
      $(\tilde{h}_i, h_i) \leftarrow$ the two contexts of direction $d(i)$ ▷ Eq. 5
      $\delta_{i,t}^{d(i)} \leftarrow \log \pi_{\bar{\theta}}(y_{i,t} \mid \tilde{h}_i, y_{i,<t}) - \log \pi_{\bar{\theta}}(y_{i,t} \mid h_i, y_{i,<t})$, $\bar{\theta} = \mathrm{sg}(\theta)$ ▷ Eq. 6
    end if
  end for
  $\mathcal{M} \leftarrow$ response tokens with $d(i) \neq \varnothing$
  ▷ Advantage reweighting
  for $(i,t) \in \mathcal{M}$ **do**
    $s_{i,t} \leftarrow 1$ if $y_{i,t}$ is an action token, ρ if a reasoning token ▷ Eq. 9
    $\lambda_{i,t} \leftarrow \lambda_k \, s_{i,t}$; $w_{i,t} \leftarrow \mathrm{clip}\big(\exp(\mathrm{sign}(\hat{A}_{i,t}) \, \mathrm{sg}(\delta_{i,t})), \, 1 \pm \varepsilon_w\big)$
    $\tilde{A}_{i,t} \leftarrow \hat{A}_{i,t}\big[(1 - \lambda_{i,t}) + \lambda_{i,t} w_{i,t}\big]$ ▷ Eq. 8; sign preserved
  end for
  $\tilde{A}_{i,t} \leftarrow \hat{A}_{i,t}$ for $(i,t) \notin \mathcal{M}$
  $\theta \leftarrow \arg\min_\theta \, \mathcal{L}_{\mathrm{RL}}(\theta)$ with $\tilde{A}$ in place of $\hat{A}$ ▷ Eq. 3
end for
return π_θ

A ALGORITHM

Algorithm 1 summarizes BEAD end to end. The augmentation table $\mathcal{T}$ is constructed once offline as described in Appendix C and then held fixed throughout training. During training, each rollout is generated in the original environment; rejected feedback is matched against $\mathcal{T}$ to obtain augmented feedback, the selected direction scores the relevant action, and the resulting log-probability gaps reweight the advantage used for policy optimization.

B BENCHMARK DETAILS AND REWARD DESIGN

All benchmarks are evaluated with their official checkers, without modifying the scoring logic. All reported results are averaged over three independent evaluation runs. We use each benchmark's official end-to-end metric for evaluation, while BFCL training uses the user-turn progress reward described below.

BFCL v4 Multi-Turn. BFCL v4 includes four stateful multi-turn tracks with 200 instances each: **Base**, where the information required for tool execution is available from the request, previous observations, or environment state; `Missing Functions`, where a required tool is unavailable until a later user turn; `Missing Parameters`, where an indispensable argument must be obtained from the user rather than inferred; and `Long Context`, where large backend states make the relevant information harder to retrieve. We train on a uniformly sampled half of `Base`, `Missing Functions`, and `Missing Parameters` and reserve the other half for evaluation. `Long Context` is never used for training.

The official multi-turn checker independently executes the policy and reference calls from the same initial environment and evaluates their outcomes after each turn. It combines a state check, which compares the resulting public backend state, with an execution-result check, which verifies that the required tool outputs are contained in the policy's accumulated execution results,

$$r_j^{\text{state}} = \mathbf{1}\big[S_j^{\pi} = S_j^{\star}\big], \qquad r_j^{\text{exec}} = \mathbf{1}\big[u_j^{\star} \subseteq_{\text{ms}} u_{\le j}^{\pi}\big], \tag{10}$$

where $\subseteq_{\text{ms}}$ denotes multiset inclusion. This outcome-based evaluation allows alternative tool-call sequences as long as they reach the required state and produce the necessary results. An instance is correct only if every turn passes the required checks.

BFCL v4 agentic tracks. We additionally evaluate on the BFCL v4 `Web Search` and `Memory` tracks. `Web Search` tests multi-hop information gathering with and without search-result snippets, whereas `Memory` tests whether an agent can store information from a preceding interaction and retrieve it for a later query. Both tracks are scored from the final non-tool response using the benchmark's deterministic answer-matching procedure. Neither track is used during training. BFCL v4 contains 200 `Web Search` and 465 `Memory` instances.

τ^2-bench. τ^2-bench evaluates conversational agents in customer-service environments where the agent must follow a domain policy while interacting with a simulated user and domain tools. We use the base split of `Retail`, `Airline`, and `Telecom`, comprising 114, 50, and 114 tasks, respectively. `Telecom` additionally gives the simulated user tools that modify the shared environment, making it a dual-control setting. Following the official configuration, we use GPT-4.1 as the user simulator.

Each task is scored by the conjunction of its task-specific binary criteria, such as consistency of the resulting environment state and required communicated information. Reference actions specify a valid target behavior but do not in general prescribe the only admissible trajectory. Following the benchmark protocol, we run three trials per task and report pass^1. More generally, with c_i successful trials out of n,

$$\widehat{\text{pass}^k} = \frac{1}{M} \sum_{i=1}^{M} \frac{\binom{c_i}{k}}{\binom{n}{k}}, \tag{11}$$

so pass^1 reduces to the mean success rate across trials.

ACEBench Agent split. We use the Agent split of ACEBench, which evaluates tool use in stateful application environments. Each language contains 20 multi-step and 30 multi-turn tasks; we evaluate both English and Chinese, for 100 instances in total. Multi-step tasks provide the full objective at the beginning, whereas multi-turn tasks reveal information through interaction with a simulated user. Following the official configuration, we use GPT-4o as the user simulator.

ACEBench evaluates the final sandbox state against the annotated target state and also provides a process metric based on intermediate milestones. We report only the end-to-end score throughout the paper. The overall score combines multi-step and multi-turn instances according to their dataset sizes.

Reward design. Following EnvTuning (Lu et al., 2026a), we use a user-turn progress reward that evaluates each user turn with two binary criteria: the correctness of the resulting environment state and the execution result. For user turn j, we define

$$r_j = r_j^{\text{state}} \cdot r_j^{\text{exec}}, \tag{12}$$

and average this signal over the N user turns,

$$R_{\text{P}} = \frac{1}{N} \sum_{j=1}^{N} r_j = \frac{1}{N} \sum_{j=1}^{N} r_j^{\text{state}} r_j^{\text{exec}} \in [0, 1]. \tag{13}$$

We use R_{P} as the reward for our main training runs, as it preserves partial progress across long multi-turn trajectories. For the reward ablation, we instead derive a sparse binary outcome reward from the same per-user-turn checks,

$$R_{\text{O}} = \prod_{j=1}^{N} r_j \in \{0, 1\}, \qquad R_{\text{O}} = \mathbf{1}\big[R_{\text{P}} = 1\big]. \tag{14}$$

Thus, the two rewards differ only in how they aggregate correctness across user turns: R_{P} retains partial progress, whereas R_{O} gives positive reward only when all user turns are completed correctly.

C MINING TOOL CONTRACTS

The augmentation table $\mathcal{T}$ is constructed once per environment before RL training. Starting from rejected calls collected from offline trajectories, we first group recurring failures into error signatures and filter out cases that do not require additional feedback. For each retained signature, an analyzer then examines the environment implementation and relevant tool interfaces to identify the implicit contract behind the rejection and generate concise guidance for resolving it. A separate validator checks the generated feedback before it is added to $\mathcal{T}$. The entire procedure is offline and uses no ground-truth task solution or oracle action sequence. The augmentations are generated automatically by an LLM-based pipeline; in practice, we use GPT-5.5 as both the analyzer and the validator.

C.1 CONSTRUCTION PROCEDURE

Grouping and triage. Rejected calls are normalized into error signatures by removing instance-specific values from the returned message and associating the resulting pattern with the tool that produced it. Calls such as "`Invalid destination airport code: SFO`" and "`Invalid destination airport code: LAX`" therefore correspond to the same signature and share one augmentation. We retain rejections that arise from explicit environment checks but whose underlying rule or resolution is not already clear to the agent. Rejections that merely restate constraints already exposed by the tool schema are discarded as uninformative, while failures caused by environment defects are excluded because no change in tool use can resolve them.

Feedback generation. For each retained signature, the analyzer is given the source code surrounding the rejection together with the relevant tool schemas and dependencies among tools or environment states. It generates a short statement that explains the contract the failed call must satisfy and, when possible, how the agent can satisfy it. We organize these contracts into four recurring types. *Parameter domain* specifies which values an argument accepts, such as a bounded integer range or a fixed set of choices. *Value format* describes how an otherwise valid value must be represented, such as using a filename rather than a full path. *Value provenance* captures cases where a valid value must come from the environment or a previous tool call, such as a system-issued booking identifier. *State precondition* describes environment state that must already hold before a tool can succeed, often established by an earlier action. The analyzer states the relevant rule and guidance without supplying task-specific values or directly solving the current task.

Validation. A separate validator admits feedback to $\mathcal{T}$ only when it satisfies five criteria. It must be *grounded*, with its claims supported by the environment implementation or visible tool interfaces; *informative*, by adding useful information beyond the original rejection and schema; *reachable*, so that any suggested resolution can actually be carried out by the agent; *task-agnostic*, remaining valid for other tasks that trigger the same rejection; and *free of leakage*, revealing neither task-specific values nor the task solution. We provide the analyzer and validator prompts in Appendix C.3 for reproducibility.

Replay check. Validation establishes that an augmentation is grounded in the environment source, but not that it actually helps the policy resolve the rejection. We therefore perform a replay-based usefulness check. Starting from the same rejected context, we run multiple subsequent rollouts under two conditions: one with the candidate augmentation and one with the original environment feedback. We retain the augmentation only if the augmented condition yields a higher estimated probability of successful continuation than the original condition. If either validation or the replay check fails, the corresponding failure reason is returned to the analyzer, which revisits the environment evidence and generates a revised augmentation. We allow up to three analysis attempts per rejection; candidates that still fail either check are discarded.

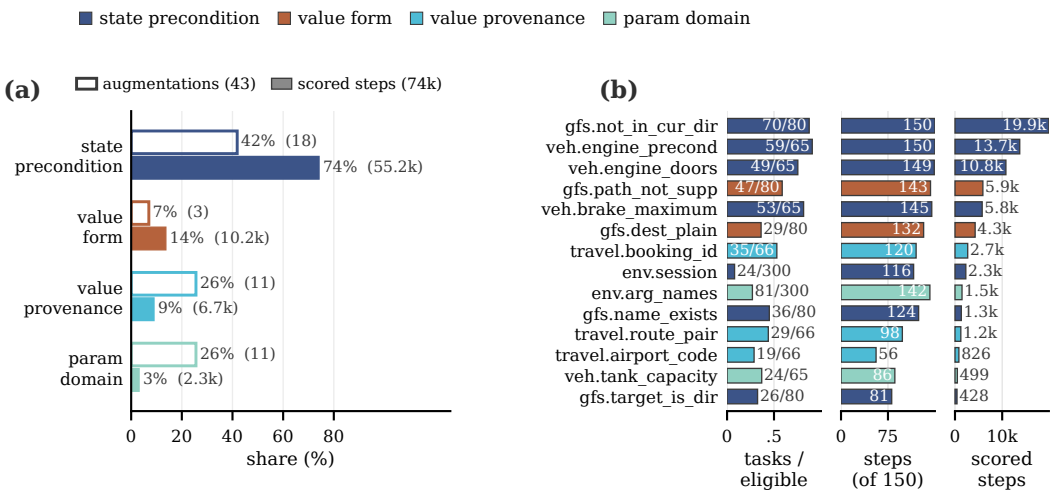


Figure 5: Additional statistics on augmentation reuse during training. All statistics are measured over one 150-step training run. **(a)** For each augmentation type, open bars show its share of the 43 augmentations triggered during training, and solid bars show its share of the resulting scored steps; counts are reported in parentheses. **(b)** The 14 augmentations with the largest numbers of scored steps, together accounting for 96% of all scored steps, colored by augmentation type. From left to right, the columns show the fraction of eligible training tasks in which each augmentation is triggered, the number of training steps in which it is triggered, and its total number of scored steps. Eligible tasks are defined by whether the task contains the API class associated with that augmentation, rather than over the full training set.

C.2 Additional Statistics on Augmentation Reuse

We provide additional statistics on how the offline augmentation set is used during training. In one 150-step run, 43 of the 56 constructed augmentations were triggered at least once. Figure 5(a) compares, for each augmentation type, its share of the triggered augmentations with its share of scored steps. The two distributions differ substantially: state-precondition augmentations account for 42% of triggered augmentations but 74% of scored steps, whereas parameter-domain augmentations account for 26% of augmentations but only 3% of scored steps. Figure 5(b) breaks this down to individual augmentations. The 14 most frequently used augmentations together account for 96% of all scored steps, with several covering most of the training tasks in which they are applicable and being triggered throughout nearly all 150 training steps. These statistics complement the reuse analysis in the main text by showing that augmentation usage is highly concentrated and that a small subset supplies a large fraction of the teacher-scored supervision during training.

C.3 Analyzer and Validator Prompts

Analyzer, system prompt

```
You improve error messages for a tool-use environment.

You get one error signature, the code that produced it, the tool schemas the agent
sees, and a sample of real trajectories that hit it. Write one short sentence to
append to the message. It ships on the message itself, so it is read by every task
that trips this guard -- not only the ones below.

# First: does the message need anything?

- Is it vague, or does it name a cause that is not the real one?
- Does it hide what comes next? The checks in the failing function run in source
  order. If clearing the one that fired runs the agent straight into the next,
  the message is hiding it.
- Does it fail to say what the constraint is, or which values are allowed?
- Does the tool's own schema promise something this code does not honour?
```

```
If none of these apply, set `need_augmentation` false and write no sentence. The
same holds when the call was fine and the environment fell over on its own state
-- a bare KeyError on a record the task loaded, an attribute error inside the
tool. No wording helps there; say so in `reasoning`.

## Then: what would your sentence teach?
Pick the one the agent has to act on differently. If two seem to fit, pick the one
     that names what the agent must do next.

- `param_domain` -- Which values this argument accepts -- a set, range or type the
     code fixes. The agent wrote a value of the right kind and the environment does
     not carry it. Nothing to look up: the sentence itself is the answer. Ticket
     priority is a whole number from 1 to 5.
- `value_provenance` -- The value cannot be authored at all; the environment issues it
     or an earlier call returns it. Say where it comes from, never what it is. A
     booking id is assigned by the travel system, so only bookings this session can
     already see may be named.
- `state_precondition` -- Some state must already hold, and another call establishes
     it. Name what must be true and what is still missing -- never what this error
     proves is already done. The brake must be held at the maximum end of its range
     before the engine starts.
- `value_form` -- The agent holds the right value in a shape this consumer does not
     take. Nothing to fetch and nothing to wait for; it has to rewrite what it already
     has. A path returned by `find` is not a name `tail` accepts.

# What the sentence must be
Six properties in three groups. A reviewer checks this same list under these same
     names, so write to it directly. One or two plain sentences, in the environment's
     own words ("filenames in the current directory", "an authenticated session")
     rather than internal variable names.

## Grounded -- is every claim traceable to what you were given?
1. `grounded` -- Every claim traceable to something you were given: the source
     (guards, tables, return values, what `_load_scenario` copies in), the tool
     schemas, the trajectories, or a disagreement between the first two. A claim does
     not have to sit in the failing function -- that booking ids can exist before this
     session booked anything is visible in what the scenario loads, not in `
     book_flight`, and it is the fact that matters. Invented requirements, defaults
     and orderings fail here, and so does any "no tool ...", "only ...", "never ...",
     "cannot ..." that is true of the excerpt and false of the class: check those
     against the whole schema, not the code in front of you. This is the most common
     failure.

## Useful -- does it save an attempt?
2. `aims_at_what_remains` -- Asks for the checks still ahead and never one the failing
     function runs *before* the one that fired -- this error existing proves those
     passed. It names the checks that follow in the same function, because a sentence
     clearing only the check that fired buys one step and another failure.
3. `reachable` -- With the original message plus this sentence, a competent agent
     could work out what to do. Fails on anything assuming information the agent
     cannot see -- a credential, an identifier it was never given. A route is a claim
     too: a lookup the sentence points at must exist in the schema and must not be
     shown raising on these records. Where the code shows no action that works,
     stating the constraint and stopping is correct; a true constraint beats a dead
     route.
4. `adds_something` -- Saves a failed attempt. A reworded copy of the original
     message, or of this tool's own schema entry, fails: the schema is already in the
     agent's system prompt. Where schema and source disagree, saying so is new.

## Safe -- does it stay inside the tool's contract, short of this task's solution?
5. `task_agnostic` -- True of the guard, not of the trajectories. The sentence ships
     on the error message itself, so every task that ever trips this guard reads it,
     including the tasks you were not shown -- the trajectories here are a sample of
     that population, not the whole of it. Name no user goal, and no file, city,
     identifier or record that appears only in the transcripts. The test: swap the
     trajectories for different tasks hitting the same error, and the sentence must
     still be right.
6. `no_leak` -- No concrete argument values, and it does not hand over the value the
     agent got wrong. Most failures here are invented data standing in for a lookup
     that was skipped, so say a lookup is needed, never what it returns. Listing
     several required conditions is not a leak.
```

```
# Examples, each tagged with the property it breaks

BAD  `task_agnostic`        "The annual report lives in the Documents folder."
                            <- true of one transcript, false of the signature
BAD  `grounded`             "No tool accepts a path."
                            <- `find` returns them, and it is in the schemas
BAD  `no_leak`              "Use booking id d184e2c0-2ebb-4f39-a525-d5e01b67dc6c."
BAD  `no_leak`              "Call list_all_airports, then use SFO and LAX."
BAD  `aims_at_what_remains` "All doors must be locked and the brake fully pressed."
                            <- the door check runs earlier, so they are locked
BAD  `aims_at_what_remains` "The brake pedal must be pressed."
                            <- a force check follows it
BAD  `reachable`            "Look the booking up in the booking history first."
                            <- that lookup raises on these records
BAD  `adds_something`       "The vehicle is not ready."

GOOD `param_domain`       "Ticket priority is a whole number from 1 to 5, where 5
                           is the highest."
GOOD `value_provenance`   "Only a booking this session can already see may be
                           referenced, and a booking that was cancelled is no
                           longer among them."
GOOD `state_precondition` "The brake pedal has to be held down at the maximum end
                           of its range before the engine will start."
GOOD `value_form`         "A name may not contain '/'. Only `find` takes a path
                           argument; every other tool resolves a plain name in the
                           current working directory."

# Output -- JSON only, no fences
{
  "reasoning": "what the check tests, where it sits, what the message leaves out, what
    a reader still cannot work out",
  "evidence": ["each claim, tagged source: / schema: / trajectory:"],
  "schema_conflict": "what the schema promises that the source does not honour" | null,
  "need_augmentation": true | false,
  "augmentation_type": "<type value>" | null,
  "hint": "<sentence to append>" | null
}
```

Analyzer, user prompt

```
# Tool schemas for {api_class}
Every tool in the class as the agent sees it. If the allowed values are already
here, say so in your reasoning -- repeating them next to the failure is useful,
inventing new ones is not. If a description promises something the code below
does not honour, that contradiction is worth more than the restatement.
```json
{class_schema}
```

Error
{error}

Tool: {tool}

Code
The failing function first, then every function that sets up the state its checks
read, `_load_scenario` among them. You do not have to hunt for the dependency: if
a function is here, it is because it can change whether this call succeeds. Read
the failing function's checks in source order -- this message came from one of
them, everything above it passed or the message could not exist, and everything
below it is still ahead of the agent. `_load_scenario` is where identifiers and
records that exist without any call having created them come from, the usual
reason a hint about where a value "comes from" turns out to be wrong. A tool that
produces the rejected kind of value may be absent here and present in the schemas
above.
```python
{related_source}
```

Trajectories that hit it
A sample of the {n_tasks} tasks, not the population. The turn's request, the calls
made, and what followed the failure. Use them to see what the agent already knew,
```

```
where it got the value it used, and whether it recovered -- and not as the subject
of the sentence, which has to hold for the tasks not shown here too.
{transcripts}

Produce the JSON.
```

Validator, system prompt

```
You review a sentence someone proposes appending to an environment's error
message. It ships on the message itself, so it is read by every task that trips
this guard, not only the ones you are shown.

Reject sentences that are made up, fitted to the trajectories, unusable,
redundant, or that point down a route that does not work.

# Check each, by name
The writer was given this same list under these same names, so a rejection can cite
    one and be acted on.

## Grounded -- is every claim traceable to what you were given?
1. `grounded` -- Every claim traceable to something you were given: the source
    (guards, tables, return values, what `_load_scenario` copies in), the tool
    schemas, the trajectories, or a disagreement between the first two. A claim does
    not have to sit in the failing function -- that booking ids can exist before this
    session booked anything is visible in what the scenario loads, not in `
    book_flight`, and it is the fact that matters. Invented requirements, defaults
    and orderings fail here, and so does any "no tool ...", "only ...", "never ...",
    "cannot ..." that is true of the excerpt and false of the class: check those
    against the whole schema, not the code in front of you. This is the most common
    failure.

## Useful -- does it save an attempt?
2. `aims_at_what_remains` -- Asks for the checks still ahead and never one the failing
    function runs *before* the one that fired -- this error existing proves those
    passed. It names the checks that follow in the same function, because a sentence
    clearing only the check that fired buys one step and another failure.
3. `reachable` -- With the original message plus this sentence, a competent agent
    could work out what to do. Fails on anything assuming information the agent
    cannot see -- a credential, an identifier it was never given. A route is a claim
    too: a lookup the sentence points at must exist in the schema and must not be
    shown raising on these records. Where the code shows no action that works,
    stating the constraint and stopping is correct; a true constraint beats a dead
    route.
4. `adds_something` -- Saves a failed attempt. A reworded copy of the original
    message, or of this tool's own schema entry, fails: the schema is already in the
    agent's system prompt. Where schema and source disagree, saying so is new.

## Safe -- does it stay inside the tool's contract, short of this task's solution?
5. `task_agnostic` -- True of the guard, not of the trajectories. The sentence ships
    on the error message itself, so every task that ever trips this guard reads it,
    including the tasks you were not shown -- the trajectories here are a sample of
    that population, not the whole of it. Name no user goal, and no file, city,
    identifier or record that appears only in the transcripts. The test: swap the
    trajectories for different tasks hitting the same error, and the sentence must
    still be right.
6. `no_leak` -- No concrete argument values, and it does not hand over the value the
    agent got wrong. Most failures here are invented data standing in for a lookup
    that was skipped, so say a lookup is needed, never what it returns. Listing
    several required conditions is not a leak.

# Output -- JSON only, no fences
{
  "reasoning": "each property in turn, by name",
  "grounded": true | false,
  "task_agnostic": true | false,
  "aims_at_what_remains": true | false,
  "reachable": true | false,
  "adds_something": true | false,
  "no_leak": true | false,
  "approved": true | false,
  "revision": "instruction for the writer, naming the property it failed" | null
}
```

```
approved is true only when every property is true.
```

Validator, user prompt

```
# Tool schemas for {api_class}
```json
{class_schema}
```

Code
```python
{related_source}
```

Original message
{error}

Proposed sentence
{hint} (type: {augmentation_type})

Claimed evidence
{evidence}

Evaluate and produce the JSON.
```

D Implementation Details

D.1 Training Parameters

All methods share the same rollout budget and optimizer settings. Each of the 150 optimization steps samples a batch of 32 tasks with a group of 16 rollouts each, and every trajectory is capped at 40 tool calls, with at most 20 calls per user request. We use no entropy regularization and no KL penalty, i.e., both coefficients are set to zero. Unless otherwise specified by a method-specific objective, all remaining optimization settings are shared across methods. Main experiments are run on 8 NVIDIA A100 80GB GPUs. Table 4 summarizes the full configuration.

Table 4: Training hyperparameters. Rollout and optimization settings are shared by all methods; the weight bound ε_w, the initial reweighting strength λ_0 and the reasoning-token scale ρ are specific to BEAD.

Category	Parameter	Value
Rollout	sampling temperature / top-p / top-k	0.99 / 0.99 / 100
	max new tokens per step	4096
	max tool calls per trajectory	40
	max tool calls per user request	20
	max sequence length	32,000
	GRPO group size	16
Optimization	learning rate	2×10^{-6}
	LR schedule	cosine
	batch size	32
	optimization steps	150
	GRPO clip ε	0.2
	weight bound ε_w	0.5
	λ_0	0.5
	reasoning token scale ρ	0.5

D.2 TRAINING SYSTEM PROMPT

Training system prompt

```
You are an expert in composing functions. You are given a question and a set of
possible functions. Based on the question, you will need to make one or more
function/tool calls to achieve the purpose.
If none of the functions can be used, point it out. If the given question lacks the
parameters required by the function, also point it out.
- When you believe the task has been completed, respond only with 'Task Completed' to
finish the current task, without adding any other content or making any tool calls.
- It is recommended to first call query tools to gather sufficient information, then
use modification tools to complete the task. Adjust actions promptly based on the
feedback from the environment, i.e., the tool results.


# 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_schemas}
</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>
```

D.3 BASELINE REIMPLEMENTATIONS

All baselines are reimplemented in ROLL (Wang et al., 2025) with the same backbone, training split, rollout budget, reward, and optimizer settings as BEAD. We preserve the defining training mechanism of each method and its original hyperparameters where applicable, while adapting benchmark-specific curriculum or privileged context to BFCL v4 without exposing any held-out evaluation information.

EnvTuning (Lu et al., 2026a) trains the acting policy directly with actionable environment feedback and progressively removes this assistance through a four-stage curriculum. Under the same 150-step budget, we first optimize format and tool-call correctness, then enable environment augmentation with the progress reward on Base, expand training to the full in-domain split (`Base`, `Missing Functions`, and `Missing Parameters`), and finally disable augmentation so that the policy finishes training under the same unaugmented environment used for evaluation.

Skill0 (Lu et al., 2026c) internalizes explicit skills by gradually withdrawing them from the policy context. We initialize it with a BFCL v4 multi-turn skill set, provide the full set during the first 50 steps, retain only a dynamically selected core set from steps 50–100, and remove skill context entirely thereafter. The core set is reselected every 10 steps using paired validation with and without skill support, retaining the skills that provide the largest validation-accuracy gains.

Skill-SD (Wang et al., 2026a) uses the current policy conditioned on skill context as a privileged self-teacher while the student always rolls out under the plain prompt. Teacher and student therefore share parameters and differ only in context; the teacher scores the student-sampled tokens, and we optimize the resulting importance-weighted reverse-KL distillation objective with the original coefficient of 0.001.

RLSD (Yang et al., 2026a) keeps the RL update direction determined by the verifier and uses the teacher–student token-level log-probability gap only to modulate its magnitude. Because BFCL provides checker states rather than a reference answer trajectory suitable as teacher context, we condition the teacher on the same BFCL skill context used by the other privileged-context baselines. We otherwise follow the original schedule: the self-distillation mixing coefficient starts at 0.5 and decays linearly to zero over the first 50 steps, after which optimization reduces to standard GRPO, with the token multiplier clipped within a radius of 0.2.



Figure 6: Per-subset validation dynamics on BFCL v4 with Qwen3-4B. Solid lines denote BEAD and dashed lines GRPO; each panel reports validation task success rate and validation progress score over 150 training steps. BEAD remains ahead on all three subsets, with larger margins on Missing Functions and Missing Parameters than on Base.

SDAR (Lu et al., 2026b) leaves the RL advantage unchanged and instead adds a token-level self-distillation term to the policy objective. The teacher again scores student-sampled tokens under privileged skill context, and the detached teacher–student log-probability gap is mapped through a sigmoid gate to weight the auxiliary likelihood term. We use the original sigmoid-gate temperature of 5.0 and auxiliary coefficient of 0.01 throughout training without annealing.

E Additional Analysis and Ablations

E.1 Per-Subset Validation Dynamics

As shown in Fig. 6, BEAD maintains higher validation task success and progress scores than GRPO across all three BFCL v4 subsets, while the margin differs substantially by subset. On Base, the advantage is relatively modest: over the final five evaluations, BEAD improves task success rate by 3.8 points and progress score by 2.3 points. The gap is larger on Missing Functions and Missing Parameters, where the corresponding improvements are 9.0/4.9 and 7.0/5.2 points in task success and progress score, respectively. These two subsets expose the policy more frequently to constrained or rejected tool interactions, providing denser environment feedback for BEAD to exploit during training.

E.2 Magnitude and Routing of Teacher Signals within the Bidirectional Run

Figure 7 shows that tbackward and forward scoring operate at markedly different signal scales within the same bidirectional run. In Fig. 7(a), the backward view produces substantially larger per-token gaps throughout training, increasing from approximately 1.08 to 1.48, whereas the forward view remains nearly flat at 0.30–0.32. The distributional view in Fig. 7(b) confirms that this is not driven by a small number of outliers: the median per-step $|\delta|$ is 1.25 for backward and 0.268 for forward, a $4.7\times$ difference, and 97.2% of backward scored steps exceed the forward median. This asymmetry is consistent with the construction of the two views. In the forward view, both student and teacher have already observed the rejection, and the teacher additionally receives the contract-level explanation; in the backward view, the teacher is instead given future environment feedback that was entirely unavailable when the rejected response was generated. The two gaps therefore measure different forms of privilege rather than two symmetric evaluations of the same signal.

The growth of the backward curve in Fig. 7(a) should not be interpreted as a comparable growth in the total amount of teacher disagreement. Over the same interval, the average backward-scored response shortens from roughly 530 to 413 tokens, while total $|\delta|$ per scored step increases only from 388 to 423. The main change is therefore signal density: a similar amount of disagreement is concentrated over fewer tokens. This distinction matters because BEAD converts δ into a token-wise

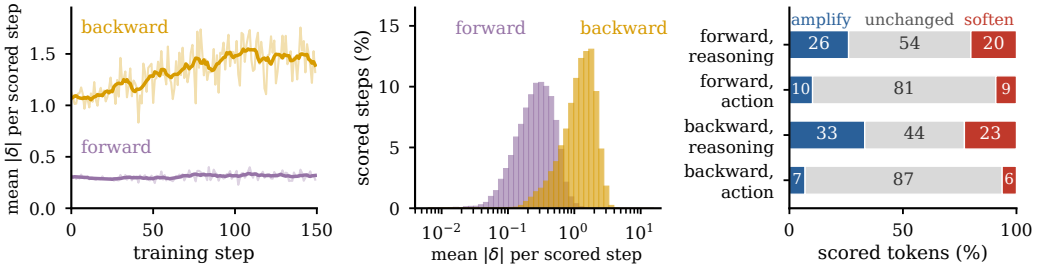


Figure 7: Magnitude, distribution, and token-level routing of the two scoring directions within the bidirectional run. All statistics are collected from the same bidirectional training run. **(a)** Left: mean token-level $|\delta|$ per scored assistant step over training; each scored step is weighted equally, light curves show raw values, and thick curves show a 9-step centered moving average. **(b)** Middle: distribution of the same per-step mean $|\delta|$, normalized separately for forward and backward views. **(c)** Right: fractions of reasoning and action tokens whose final reweighting factor amplifies ($w > 1$), leaves approximately unchanged ($|w - 1| \leq 5 \times 10^{-6}$), or softens ($w < 1$) the original advantage.

reweighting factor. Fig. 7(c) shows where that reweighting actually lands. On reasoning tokens, 46% of forward tokens and 56% of backward tokens are changed, compared with only 19% and 13% of action tokens, respectively. Thus, although the teacher ultimately supervises tool-use behavior, most of the reweighting acts through the reasoning stream rather than directly rewriting the emitted call. The next analysis decomposes this effect further and shows that forward and backward distribute their action- and reasoning-level supervision in systematically different ways.

E.3 Token-Level Complementarity across Directional Ablations

Figure 8 compares the token-level reweighting patterns of the forward-only, backward-only, and bidirectional runs from the directional ablation study. We separate three quantities that are otherwise conflated in a single average reweighting statistic: the fraction of selected tokens whose weight actually changes, the pre-clipping teacher–student gap $|\delta|$ on those changed tokens, and the final deviation $|w - 1|$ that enters advantage reweighting. Together, these quantities distinguish how broadly a teacher view acts from how strongly it acts where it does. On action tokens, the forward-only and backward-only runs exhibit a clear coverage–strength trade-off: forward changes more tokens, whereas backward changes fewer tokens but produces substantially larger $|\delta|$ on the tokens it affects.

This difference also explains the reversal across the three action-token panels. Although backward produces the larger conditional $|\delta|$, its stronger gaps are more likely to saturate under the clipping in equation 8, so part of their additional magnitude is removed before entering the RL update. Forward instead distributes smaller gaps over a larger fraction of action tokens, allowing more of its signal to survive the clipping transformation. As a result, the ordering reverses again in the final $|w - 1|$: forward produces the larger effective perturbation on action tokens despite having weaker per-token disagreement before clipping. The three panels are therefore not contradictory; they reveal two distinct supervision shapes, with forward acting broadly but moderately and backward acting sparsely but sharply on the action stream.

Reasoning tokens show a different pattern. Across coverage, conditional $|\delta|$, and final $|w - 1|$, backward remains consistently stronger than forward, indicating that hindsight feedback acts more broadly and more strongly on the reasoning that precedes a rejected call. Taken together, the two rows suggest that the directional views emphasize different token-level learning channels: forward supervision is relatively more prominent on the recovery action, whereas backward supervision is concentrated more heavily on the reasoning that produced the rejected call. The bidirectional run retains both forms of signal, providing a token-level account of the complementarity observed in the main directional ablation, where removing either view reduces performance.

E.4 Ablations on Feedback Budget and Reasoning-Token Scaling

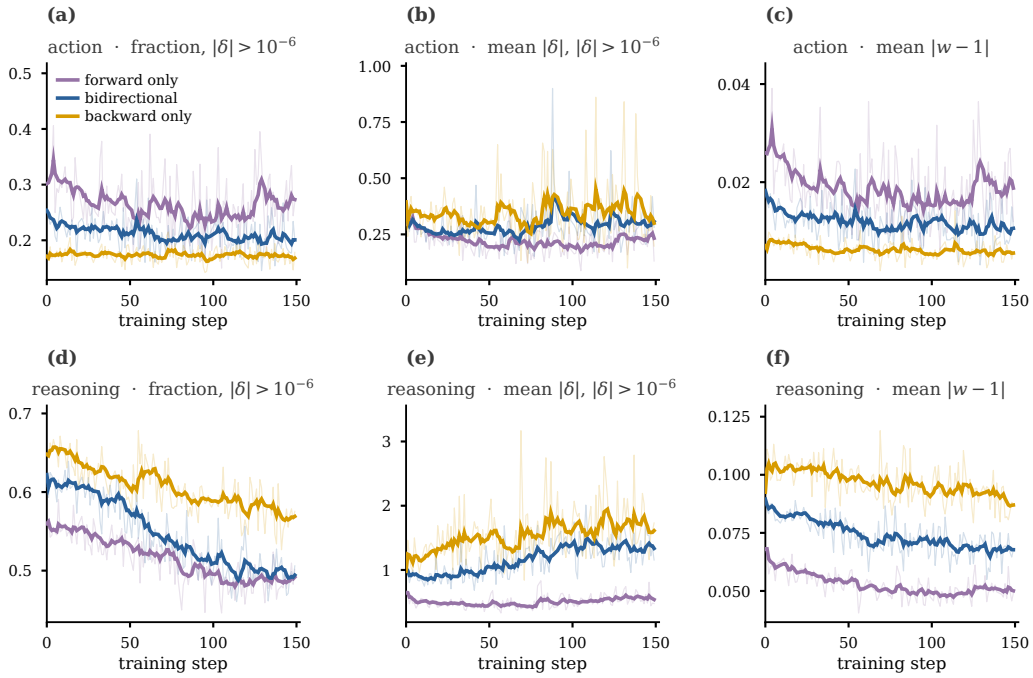


Figure 8: Coverage, strength, and effective reweighting across the forward-only, backward-only, and bidirectional runs. The three curves correspond to the directional ablation runs used in the main ablation study. Rows separate action tokens **(a–c)** from reasoning tokens **(d–f)**. Left **(a, d)**: fraction of selected tokens whose final weight changes, $\Pr(|w-1| > 10^{-6})$, measuring supervision coverage. Middle **(b, e)**: mean pre-clipping $|\delta|$ conditioned on the token being changed, measuring local teacher–student disagreement. Right **(c, f)**: mean $|w-1|$ over all selected tokens, measuring the effective perturbation that enters advantage reweighting. Light curves show raw training statistics and thick curves their smoothed trajectories.

We first vary the per-trajectory feedback budget B, which controls how many distinct augmented feedback items are selected for teacher scoring. As shown in Fig. 9, performance improves steadily as the budget increases from $B = 2$ to $B = 8$, with the BFCL v4 in-domain average rising from 46.00 to 51.56. Increasing the budget further to $B = 16$ yields only a small additional gain to 52.00. This suggests that most of the benefit is already captured at $B = 8$, which we adopt in the main experiments to limit additional teacher computation while retaining broad feedback coverage.

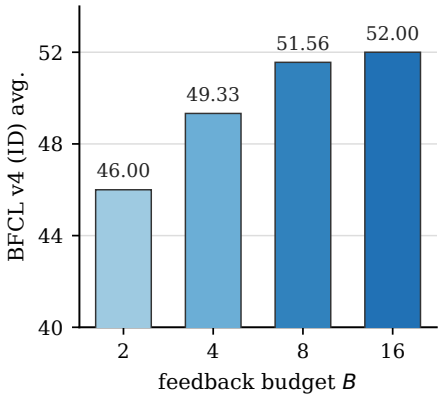


Figure 9: Average BFCL v4 multi-turn performance as a function of the per-trajectory augmented-feedback budget B. Per-subset numbers are reported in Table 5.

We next ablate the reasoning-token scale ρ. The default setting $\rho = 0.5$ downweights reasoning tokens relative to executable action tokens, whereas $\rho = 1$ applies the same reweighting strength to both. As shown in Table 5, removing this downweighting decreases the average score from 51.56 to 48.89, with consistent drops across `Base`, `Missing Functions`, and `Missing Parameters`. This supports allocating stronger teacher reweighting to action tokens, where the augmented feedback is more directly grounded, rather than allowing long reasoning spans to receive equal weight despite weaker attribution.

Table 5: **Ablation on the reasoning-token scale ρ.** Δ Avg. reports the change in average performance relative to the default $\rho = 0.5$. All runs use Qwen3-4B.

	BFCL v4 multi-turn				
ρ	**Base**	**Miss Func**	**Miss Param**	**Avg.**	**Δ Avg.**
0.5 *(default)*	57.33	52.00	45.33	51.56	–
1.0	55.00	48.00	43.67	48.89	-2.67

F Case Study

F.1 Example of Token-Level Supervision

Figure 10 opens up the token-level supervision on a single training trajectory. The user asks for the distance from Rivermist to San Francisco. After seeing that `estimate_distance` requires zip codes, the policy incorrectly reasons that "the tools don't include a function to get a city's zipcode," even though `get_zipcode_based_on_city` is available in the same API suite and had already been used earlier in the trajectory. It therefore guesses `94101`, the call is rejected with only `distance not found in database`, and the policy subsequently misdiagnoses the failure as evidence that the environment lacks the requested distance information. The augmented feedback supplies the missing environment knowledge: distances are indexed by zip-code pairs, guessed city identifiers need not correspond to valid entries, and the available API can retrieve a city's zip code.

The black boxes highlight how this misdiagnosis develops across the two scored actions. In panel (a), box ❶ marks the internal justification that enables the bad call: because the policy believes no zip-code lookup tool exists, guessing appears reasonable. In panel (b), this local excuse becomes a broader conclusion. Box ❷ states that there is no way to retrieve distance information beyond `estimate_distance`, turning one failed lookup into a claim about the capability of the entire tool set. Box ❸ then externalizes that conclusion to the user: "The provided tools do not have access to distance data between cities." The token-level log-probability gaps concentrate on these statements, showing that the augmented feedback does not merely react to the rejected call itself; it targets the chain of unsupported beliefs through which a local tool-use mistake becomes a persistent misunderstanding of the environment.

F.2 Behavioral Comparison with GRPO

Figure 11 provides qualitative examples of how BEAD changes the agent's interaction with tool-use environments. Across the four cases, the main difference is not simply that BEAD produces fewer rejected calls. Rather, it shows a more accurate understanding of the tool contract and of how the environment state constrains subsequent actions. This is particularly clear in Figure 11(a): after changing into `project`, GRPO still interprets `find(path=...)` as if the path were resolved from the parent directory, and repeats the same rejected call even after listing the target file. BEAD instead recognizes that the path is relative to the current working directory and completes the task. Figure 11(b) shows a complementary failure mode. When the user leaves the destination unspecified, GRPO invents one and then treats the resulting rejection as a formatting problem; BEAD avoids committing to an unsupported choice and waits for the missing information.

The remaining cases highlight rejection diagnosis and recovery. In Figure 11(c), BEAD consistently uses the environment feedback to revise its subsequent behavior: it corrects the reminder title, queries the available products after a product-name rejection, and clears inbox capacity before retrying the message. GRPO, by contrast, interprets "No reminder found" as evidence that the reminder itself does not exist and fails to recover the intended task. This case illustrates that effective tool use depends not only on recognizing that a call has been rejected, but also on identifying the underlying cause and translating that diagnosis into the next action.

Figure 11(d) further illustrates why understanding environment dynamics matters in stateful tool use. Both return and exchange operations require the order to be in the delivered state, but executing either operation changes that state. GRPO recognizes the initial order state yet issues both calls together, causing the second operation to become invalid after the first succeeds. BEAD instead exposes the mutual exclusion to the user, follows the stated preference for exchange, and completes

Trajectory Progress Score: 0.50 Trajectory Advantage: -1.85

rejection feedback

```
Function Call {'name': 'estimate_distance', 'arguments':
{'cityA': '83214', 'cityB': '94101'}} Succeeded. Result:
{'error': 'distance not found in database.'}
```

augmented feedback
(teacher privileged context)

```
Future environment feedback:
Function Call {'name': 'estimate_distance', 'arguments':
{'cityA': '83214', 'cityB': '94101'}} Succeeded. Result:
{'error': 'distance not found in database.'} Distances are
stored against pairs of zip codes, not city names, and only for
the pairs this API records. A city name, or a zip code that was
assumed rather than looked up, never resolves; the API can
supply the zip code for a city.
```

(a) backward rescore

<think> Okay, the user wants to know the distance from Rivermist to San Francisco.… There's the estimate_distance function which requires cityA and cityB zipcodes. Since I already have River

❶ mist's zipcode as 83214, I need to find San Francisco's zipcode. But wait, the tools don't include a function to get a city's zipcode. Hmm, maybe the user expects me to know San Francisco's zipcode. I remember that San Francisco's zipcode is 94101.… The function will return the distance in km.…

</think>

<tool_call> {"name": "estimate_distance", "arguments": {"cityA": "83214", "cityB": "94101"}}

</tool_call>

(b) forward rescore

<think> … I called the estimate_distance function with those zipcodes, but it returned an error saying the distance wasn't found in the database. Hmm, maybe the system doesn't have that data.

❷ Since the tools provided don't include a way to retrieve distance information beyond the estimate_distance function, and that failed, I can't proceed further.…

</think>

❸ The provided tools do not have access to distance data between cities. The `estimate_distance` function failed to retrieve the distance, so I cannot determine it with the available functionality.

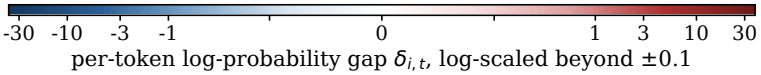
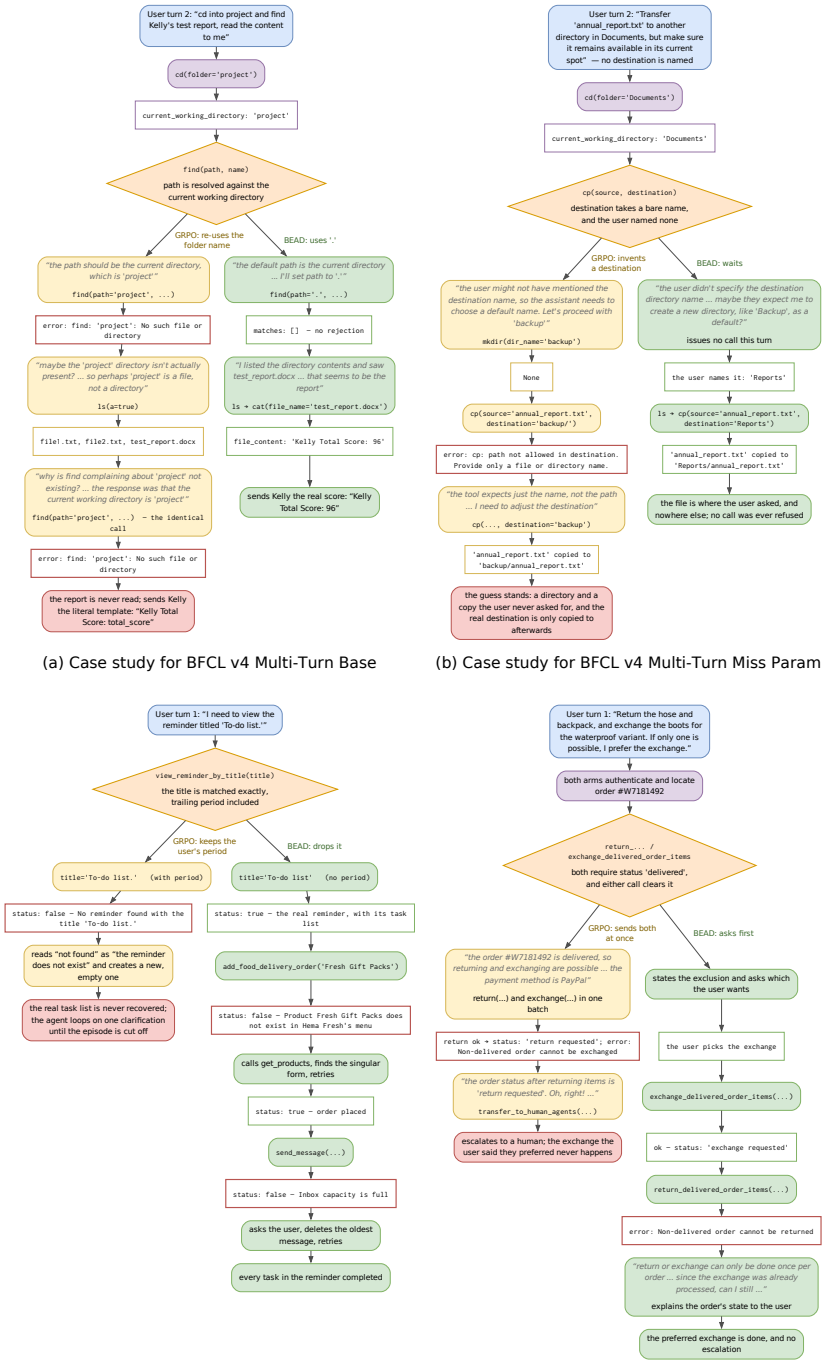


Figure 10: Example of Token-Level Supervision. The same augmented feedback is used to score the rejected call and the subsequent response; the highlighted spans show how an incorrect assumption about tool availability is first used to justify the call, then generalized into a conclusion about the tool set, and finally stated to the user as an environmental limitation.

the preferred operation before the state changes. Together, these cases suggest that BEAD improves not only tool-call selection, but also the agent's ability to diagnose rejections, reason about latent tool constraints, and anticipate how actions change the environment for what comes next.



(a) Case study for BFCL v4 Multi-Turn Base

(b) Case study for BFCL v4 Multi-Turn Miss Param

(c) Case study for ACEBench-Agent

(d) Case study for τ^2-bench

Figure 11: Qualitative comparison of GRPO and BEAD across four multi-turn tool-use settings. Each panel contrasts the two policies on the same task instance and highlights the tool or environment constraint at which their behaviors diverge. Diamonds summarize the relevant tool contract or state constraint; rounded boxes show agent turns, with italicized text giving abbreviated reasoning and the text below showing the issued tool call; rectangular boxes show environment returns, with red borders marking rejected calls. The examples illustrate differences in relative-path handling, missing-parameter decisions, rejection recovery, and state-dependent tool interactions.