

ANTE: Anticipate, Then Execute – Learning Internal World Models for Coding Agents from Verified Beliefs

Yi Ge^{1,*}, Kangrui Ruan^{2,†}, Jifan Chen², Manish Prajapat², Christian Bock², Xiaoyang Wang², Rein Houthoofd², Aritra Sengupta², Varun Kumar²

¹University of Maryland, College Park, ²AWS AI Labs

*Work done during an internship at AWS AI Labs, [†]Corresponding author

Rapid advances in large language models have made software engineering agents capable of tackling complex coding tasks. Within a containerized environment, a coding agent receives tool responses or execution results, and determines its later actions by those earlier results. However, most existing coding agents never look ahead: their policies do not anticipate the consequences of actions, and the terminal reward barely attributes a failure to certain earlier steps, so an ill-judged edit can corrupt the repository state irreversibly before any signal arrives. Recent code world models aim to address these challenges, but they are either trained and then frozen before reinforcement learning (RL), or reduced to an auxiliary next-token loss on the environment’s outputs. We present ANTE (Anticipate, Then Execute), an agentic RL framework that verifies the policy’s pre-action predictions by real execution and turns them into per-step credit for its actions and trust-or-abstain decisions. With an internal belief stated before each action, Belief-Anchored Policy Optimization (BAPO) turns the settled predictions into per-step advantages by anchoring credit on the policy’s own forecasts. To turn the trained forecasts into better actions at inference, thereby facilitating effective test-time scaling, ANTE-TTO (test-time optimization) selects among candidate actions without an external scorer. ANTE raises Qwen3-32B from 25.0% to 57.2% pass@1 on SWE-bench Verified, 6.2% above a matched GRPO baseline trained without the belief state; its world model’s accuracy climbs from 44% to 69% during RL, and the learned foresight capability transfers across harnesses and out-of-distribution domains. In summary, we aim to build a code world model that knows what has happened, what holds now, and what the next action will achieve.

Author E-mails: yige@umd.edu, kangruir@amazon.com

Project Page: <https://ante-agent.com>



1 Introduction

Large language models (LLMs) now act as software engineering agents: given a repository issue or a terminal task, they edit, run, and iterate inside a container until tests validate the result (Jimenez et al., 2024; Pan et al., 2025; Jain et al., 2025). Open agents are trained by supervised fine-tuning (SFT) on executed trajectories (Yang et al., 2025b) and by reinforcement learning (RL) against that terminal test (Song et al., 2026), at growing scale. Both SFT and RL are trained on tool-use trajectories, in which the agent observes each tool result and uses it to choose the next action (Luo et al., 2025; Yang et al., 2025c). However, outcome-based RL rewards only the final task result (Shao et al., 2024; Wang et al., 2025c; Ding et al., 2026); it does not directly train the agent to predict what each action will do. Without anticipating an action’s consequences, an agent may solve the immediate problem while introducing errors elsewhere in the repository. Figure 1 illustrates this limitation: both the reactive agent (thinking-only) and the retrospective agent (thinking with memory) submit an edit that passes the target test but breaks a previously passing test. The memory records what happened, but neither agent anticipates the edit’s side effects before acting.

To anticipate such failures, an agent needs to predict an action’s outcome before executing it. A world model provides this prediction, which can then be checked against the execution result, such as whether a test will pass. The observed outcome then serves as a training target, helping the world model make more accurate predictions. Existing approaches learn from execution traces or auxiliary observation losses (Meta FAIR

CodeGen Team, 2025; Shrivastava et al., 2026; Lu et al., 2026), or refine simulated plans (Yu et al., 2025b), but do not turn verified forecasts into per-step policy credit. Other approaches rely on learned simulators or language-model judges to evaluate predictions (Sun et al., 2026; Wang et al., 2025b). Execution-verified reasoning has been explored within individual programs (Ni et al., 2024; Wang et al., 2026; Tang et al., 2026a), rather than repository-level agent trajectories. The remaining gap is to combine explicit forecasts, verification by real execution, and per-step credit within agentic RL.

We introduce **ANTE (Anticipate, Then Execute)**, an agentic RL framework that trains coding agents to predict action outcomes and learn from execution feedback. Before each action, the policy writes a *belief state* containing a brief state summary, an outcome prediction, and a *trust* or *abstain* decision. Our code world model maintains an internal belief about what the agent has established, what the code does now, and what the next command will change. During RL, ANTE utilizes Belief-Anchored Policy Optimization (BAPO), which compares steps across rollouts that predict the same outcome and assigns per-step credit based on their verified forecasts and trust decisions. At inference, ANTE-TTO samples alternative actions when the policy abstains or predicts failure, ranks the candidates by their own forecasts, and executes the highest-ranked action without an external scorer.

Figure 1 illustrates how this helps: unlike the reactive and retrospective agents, ANTE anticipates that the proposed edit would break other tests, chooses an alternative fix, and confirms that the full test suite passes. This example illustrates the behavior we seek: anticipating an action’s consequences rather than reacting to them afterward.

Our main contributions are: ❶ To our knowledge, ANTE is the pioneering agentic RL framework in the code domain that verifies the policy’s explicit forecasts against real execution throughout RL. These verified forecasts provide step-level supervision for learning an internal world model within the policy. ❷ To turn verified predictions into per-step credit, we propose BAPO. It groups steps across rollouts by their predicted outcomes and compares their execution feedback, allowing individual steps to receive credit beyond the final task reward. ❸ We also propose ANTE-TTO, which uses the learned forecasts to choose among candidate actions before execution at inference time. It ranks candidates by their predicted outcomes and executes only the highest-ranked action. At 7B scale, it improves task success rate with approximately 31% additional inference tokens over a single greedy attempt. ❹ Our experiments show that ANTE improves both prediction accuracy and task performance across tasks and harnesses. ANTE-32B reaches 57.2% on SWE-bench Verified (Jimenez et al., 2024), outperforming all compared open-weight agents above 30B. ANTE-9B reaches 24.7% on Terminal-Bench 2.0 (Merrill et al., 2026). In addition, across eleven out-of-distribution benchmarks, ANTE-32B improves average performance by 6.2 percentage points over a matched RL baseline trained without the belief state.

2 Related Work

World models for coding agents. Code world models could help coding agents anticipate the consequences of their actions. CWM (Meta FAIR CodeGen Team, 2025) learns to predict program execution during mid-training, while its RL stage does not reward or check these predictions. SWE-World and PatchWorld (Sun et al., 2026; Bai et al., 2026) instead model the environment outside the policy, as a learned simulator or an

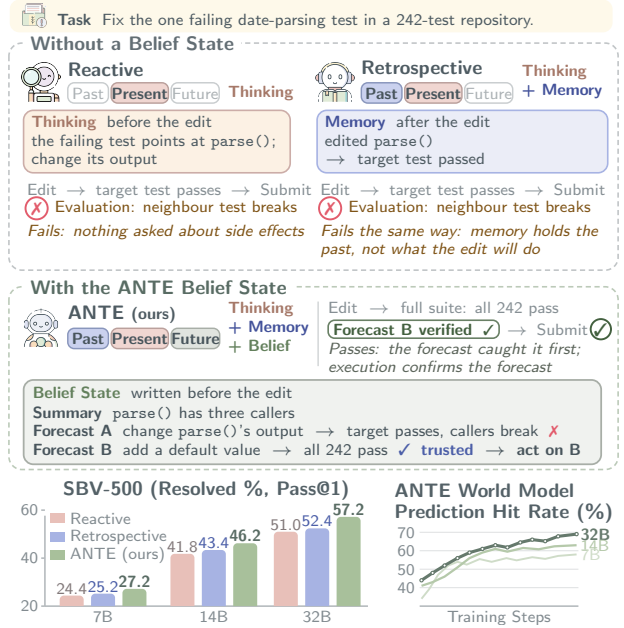


Figure 1 Anticipation, verified by execution. Top: the same task with and without a belief state; only ANTE forecasts the edit’s effect and lets execution settle it before committing. Bottom: under RL, resolve rate and prediction hit rate rise together.

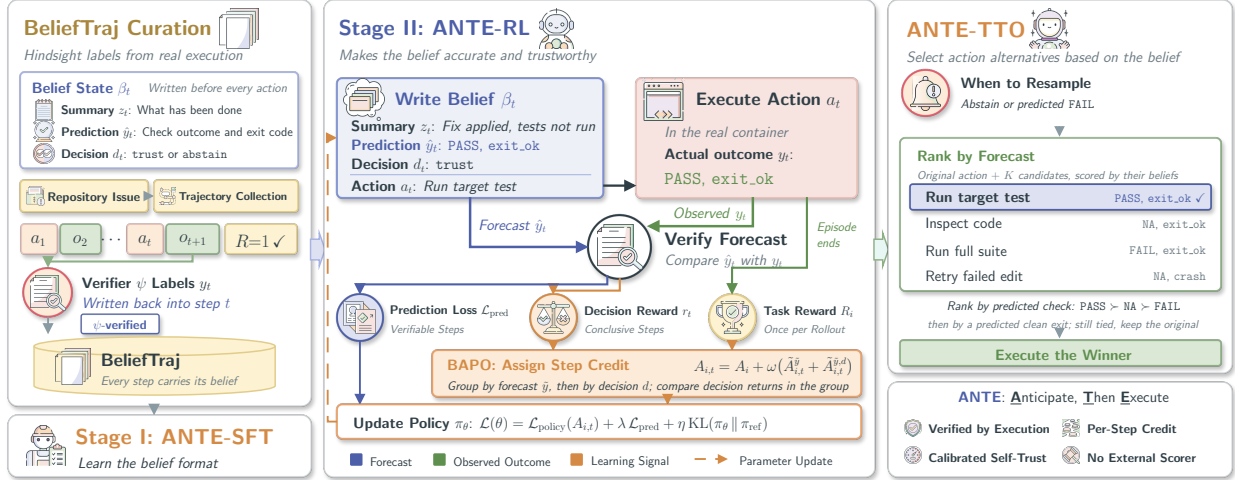


Figure 2 Overview of ANTE. In Stage I, the model learns the belief state from execution-labeled BeliefTraj; Stage II verifies every forecast against execution and turns it into a prediction loss, a decision reward, and BAPO step credit; ANTE-TTO resamples at an abstention or a predicted FAIL and executes the best-forecast candidate.

executable program. ECHO and PaW (Shrivastava et al., 2026; Lu et al., 2026) add an auxiliary loss on the environment’s outputs during RL, yet the policy still acts without first stating a forecast. In contrast, ANTE has the policy forecast each action’s outcome before acting, verifies the forecast by real execution throughout RL, and uses it for per-step credit and, at test time, for ranking candidate actions.

Structured beliefs in agentic RL. Prior work rewards an agent’s beliefs during RL by comparing them with the true state, through a language-model judge (Wang et al., 2025b; Singh et al., 2026) or embedding similarity (Yu et al., 2026). However, such comparisons only approximate correctness, and judge-based rewards are prone to reward hacking (Wang et al., 2025b; Yu et al., 2026). T3 (Zou et al., 2026) keeps the belief implicit and truncates rollouts once proxy signals show that it has stalled, leaving the belief itself unsupervised. ANTE grounds the belief in verifiable action outcomes, which makes the supervision more robust: real execution checks each forecast and the decision to trust it.

Credit assignment and execution-grounded reasoning. Deciding which steps across rollouts are comparable is central to per-step credit assignment. GiGPO (Feng et al., 2025) compares actions taken from the same environment state, which requires identical states to recur across rollouts. ReBel (Tang et al., 2026b) groups steps whose beliefs agree on a set of predefined predicates, while PRIME (Cui et al., 2025) instead learns an implicit process reward model that gives dense rewards. Code-reasoning methods use execution traces within individual programs rather than repository-level agent trajectories (Wang et al., 2026; Tang et al., 2026a; Ni et al., 2024). Our proposed BAPO also compares steps that make the same forecast in different rollouts of the task.

3 ANTE: Learning to Predict Before Acting

3.1 Problem Formulation

We formalize a coding agent working on a task inside its container as a partially-observable Markov decision process (POMDP) (Åström, 1965; Kaelbling et al., 1998) whose latent state includes the repository (Appendix A). Figure 2 provides an overview of the ANTE pipeline. The agent can predict the outcome y_t of the check that its next action runs, and execution verifies that prediction. A deterministic verifier ψ reads it from the execution result as a pair in $\mathcal{Y} = \mathcal{Y}^{\text{tst}} \times \{0, 1\}$. The first field is the check’s result, PASS, FAIL, or NA; the second records whether the command exited cleanly. ANTE’s belief is a point forecast of this outcome, written before the tool call.

Belief state. Before every action, the policy explicitly articulates what it believes. The belief state $\beta_{i,t} = (z_{i,t}, \hat{y}_{i,t}, d_{i,t})$ has three parts: a one-sentence summary $z_{i,t}$, a forecast $\hat{y}_{i,t} = (\hat{y}_{i,t}^{\text{tst}}, \hat{y}_{i,t}^{\text{exe}}) \in \mathcal{Y}$, and a decision $d_{i,t} \in \{\text{TRUST}, \text{ABSTAIN}\}$. The summary $z_{i,t}$ records what is established and what is broken. The forecast takes its values in the outcome space of the verifier, so that it is a statement the harness can confirm or refute. The decision $d_{i,t}$ is a two-level certainty label in the sense of selective prediction (Davidov et al., 2026; Wu et al., 2026a). TRUST commits to the forecast. ABSTAIN still writes the forecast and still takes the action, but does not commit to it. Intuitively, the decision is needed because prediction accuracy alone would reward guessing wherever the repository state is underdetermined. A confident forecast of failure is a legitimate TRUST. Within one autoregressive pass the step message factorizes into reasoning, summary, prediction, decision, and action factors,

$$\begin{aligned} \pi_{\theta}(u_{i,t} \mid h_{i,t}) &= \pi_{\theta}(v_{i,t} \mid h_{i,t}) \pi_{\theta}(z_{i,t} \mid h_{i,t}, v_{i,t}) \pi_{\theta}(\hat{y}_{i,t} \mid c_{i,t}) \\ &\times \pi_{\theta}(d_{i,t} \mid c_{i,t}, \hat{y}_{i,t}) \pi_{\theta}(a_{i,t} \mid h_{i,t}, v_{i,t}, \beta_{i,t}), \end{aligned} \quad (1)$$

where $h_{i,t}$, $v_{i,t}$, and $a_{i,t}$ are the history, the reasoning, and the tool call, respectively, $c_{i,t} := (h_{i,t}, v_{i,t}, z_{i,t})$ is the context in which the forecast is written, and $\pi_{\theta}(y \mid c_{i,t})$ is the probability the prediction factor assigns to outcome y .

Verifiable and conclusive steps. A forecast can be scored only when the action runs something the verifier can read. Before execution, a deterministic action-type map $\chi : \mathcal{A} \rightarrow \{\text{EPI}, \text{PRAG}\}$ sorts the steps. A step is *epistemic* if its action runs a check the verifier can evaluate, such as a target test or a reproduction script, and *pragmatic* otherwise. After execution, the verifier $\psi : \mathcal{O} \times \{\text{EPI}, \text{PRAG}\} \rightarrow \mathcal{Y} \cup \{\perp\}$ evaluates the step. The check field $y_{i,t}^{\text{tst}}$ records the result of the check that an epistemic step ran, or NA when the response does not determine one. The exit field $y_{i,t}^{\text{exe}}$ records whether the command ran cleanly. The realized outcome is $y_{i,t} := \psi(o_{i,t+1}, \chi(a_{i,t}))$. A step is verifiable when its belief block parses and ψ returns a value ($m_{i,t} = 1$). It is *conclusive* when in addition $y_{i,t}^{\text{tst}} \in \{\text{PASS}, \text{FAIL}\}$; on repository issues these are the verifiable epistemic steps whose check produced a result. We write $\mathcal{C}_i := \{t \in [T_i] : m_{i,t} = 1, y_{i,t}^{\text{tst}} \in \{\text{PASS}, \text{FAIL}\}\}$ for the conclusive steps of a rollout and $\mathcal{C} := \{(i, t) : t \in \mathcal{C}_i\}$ for those of the task.

3.2 Stage I (ANTE-SFT): Supervised Fine-Tuning on BeliefTraj

Our training procedure consists of two stages: supervised fine-tuning (SFT) followed by reinforcement learning (RL). In Stage I, we teach the policy to generate structured belief states alongside its reasoning and actions through SFT on BeliefTraj, a corpus of solving trajectories collected by a capable agent in the execution environment. Each step is annotated with a belief state $\beta_t = (z_t, \hat{y}_t, d_t)$. The forecast $\hat{y}_t$ is the outcome the verifier ψ reads from the logged next observation. The decision d_t is a label assigned when the corpus is built. Stage II subsequently refines the policy’s trust/abstain behavior through decision rewards. Construction, decision labeling, the leakage filter, and statistics are in Appendix D. Stage I jointly optimizes all policy factors in Eq. 1 by maximum likelihood on the corpus $\mathcal{D}_{\text{BT}}$, $\mathcal{L}_{\text{ANTE-SFT}}(\theta) = -\mathbb{E}_{\tau \sim \mathcal{D}_{\text{BT}}} \sum_t \log \pi_{\theta}(u_t \mid h_t)$; observation and prompt tokens carry no loss. After Stage I, belief is part of the policy’s normal output, and the prediction factor is correct in hindsight on corpus histories. Building on prior work on imitation learning and SFT (Gangwani et al., 2020; Ruan et al., 2024; Chu et al., 2025), we formulate Stage I as belief-augmented imitation in a POMDP and derive the following bound:

Proposition 1 (Belief-augmented imitation bound). *Let P_E be the law of BeliefTraj trajectories over the corpus task distribution, and let π_{θ} act in the real environment on the same tasks. Write J for the task success probability, T for the step budget, and $H_E := \sum_{t=1}^T \mathbb{E}_{P_E}[H(P_E(\cdot \mid h_t))]$ for the conditional entropy of the demonstration process’s step messages $u_t = (v_t, \beta_t, a_t)$, reasoning included, $\varepsilon_{\text{SFT}}(\theta) := \mathcal{L}_{\text{ANTE-SFT}}(\theta) - H_E \geq 0$ for the excess Stage I loss, and $\Lambda := \sum_{t=1}^T \mathbb{E}_{P_E}[\text{TV}(P_E(o_{t+1} \mid h_t, u_t), P_{\text{env}}(o_{t+1} \mid h_t, a_t))]$ for the non-causal part of the demonstration process. Then*

$$J(P_E) - J(\pi_{\theta}) \leq \sqrt{T \varepsilon_{\text{SFT}}(\theta)/2} + \Lambda. \quad (2)$$

Proposition 1 states what Stage I guarantees. Minimizing the cross-entropy drives the first term down, so the fine-tuned policy approaches the success rate of the demonstrations up to Λ . This remainder comes from the labels written in hindsight, which imitation cannot recover. Stage II addresses the hindsight-induced mismatch captured by Λ .

3.3 Stage II (ANTE-RL): Belief-Anchored Policy Optimization

In Stage II, the policy learns from its own rollouts in the execution environment, using execution feedback to refine its forecasts and decisions. These rollouts provide step-level execution outcomes and episode-level task rewards, which support a joint objective combining prediction supervision with belief-anchored policy optimization (BAPO).

RL rollouts. For each training task $x \sim \mathcal{D}$, the policy π_θ produces N rollouts $\tau_1, \dots, \tau_N \sim \pi_\theta(\cdot \mid x)$. At step t of rollout i , the policy reads the history $h_{i,t} := (x, u_{i,1}, o_{i,2}, \dots, u_{i,t-1}, o_{i,t})$ and emits one message $u_{i,t} = \langle v_{i,t}, \beta_{i,t}, a_{i,t} \rangle$: its reasoning, a belief state, and one tool call. The environment replies with the observation $o_{i,t+1}$. A rollout ends after T_i steps.

Given these rollouts, the prediction loss supervises forecasts with execution-derived labels, while the decision reward scores trust/abstain decisions on conclusive steps. The policy-gradient update supplements episode-level task credit with belief-anchored adjustments on eligible steps, computed from decision returns within forecast and forecast–decision groups. The decision reward and belief-anchored advantage construction together form BAPO. The policy gradient trains the reasoning, decision, and action tokens of Eq. 1; the prediction loss alone trains the two prediction tokens; the summary tokens receive no gradient, since nothing checks a summary. The task reward, received once per rollout, is $R_i := \mathbf{1}[\tau_i \text{ resolves } x] + r_i^{\text{fmt}}$ with a bounded bookkeeping term for format and length (Appendix F). To motivate Stage II’s prediction supervision, we consider a selector that ranks candidate actions by their predicted expected utility. The following proposition bounds its one-step selection regret in terms of candidate-specific forecast divergences.

Proposition 2 (Prediction-to-decision regret bound). *Let candidates $k = 0, \dots, K$ at one step have outcome laws p_k under execution and belief predictives q_k , with $\delta_k := \text{KL}(p_k \parallel q_k)$ the divergence of each forecast from its executed outcome law. For a score $u : \mathcal{Y} \rightarrow [0, U]$, let k^* maximize the executed value $v_k := \mathbb{E}_{p_k}[u]$ and $\hat{k}$ the believed value $\hat{v}_k := \mathbb{E}_{q_k}[u]$. Then*

$$v_{k^*} - v_{\hat{k}} \leq U \left(\sqrt{\delta_{k^*}/2} + \sqrt{\delta_{\hat{k}}/2} \right) \leq U \sqrt{2 \max_k \delta_k}. \quad (3)$$

Proposition 2 shows that smaller candidate-specific forecast divergences tighten an upper bound on selection regret. Detailed proofs can be found in Appendix C.

Execution-grounded prediction loss. For a fixed conditional outcome distribution, expected forecast cross-entropy equals KL divergence plus an entropy term independent of the predictive distribution. This motivates supervising forecasts with the realized outcomes of the policy’s own actions. The prediction factor occupies the value-token positions $\mathcal{V}_{i,t}$ of a step. The reasoning, decision, and action factors occupy the positions $\mathcal{K}_{i,t}$. Both losses are averaged over the same token count $Z := \sum_{i,t} |\mathcal{K}_{i,t}|$. Every verifiable step receives a cross-entropy against the realized outcome at its value positions,

$$\mathcal{L}_{\text{pred}}(\theta) = -\mathbb{E}_{x \sim \mathcal{D}} \mathbb{E}_{\{\tau_i\}_{i=1}^N \sim \pi_\theta(\cdot \mid x)} \left[\frac{1}{Z} \sum_{i=1}^N \sum_{t=1}^{T_i} m_{i,t} \log \pi_\theta(y_{i,t} \mid c_{i,t}) \right]. \quad (4)$$

Decision reward. The decision reward encourages accurate forecast commitments and penalizes incorrect ones. On conclusive steps, the scoring rule σ (Eq. 19) assigns a reward of $+\alpha$ to a trusted forecast that is correct in both fields with a clean exit, and a penalty of $-\gamma$ to an incorrect trusted forecast. Abstention incurs a fixed cost κ , contributing $-\kappa$ to the reward. We define the decision reward of step t as $r_{i,t} := \sigma(d_{i,t}, \hat{y}_{i,t}, y_{i,t})$ for $t \in \mathcal{C}_i$ and $r_{i,t} := 0$ otherwise, and its decision return as $W_{i,t} := \sum_{s \geq t} r_{i,s}$, the sum of the rewards from step t on. We set $\gamma = \alpha q^*/(1 - q^*)$, so that trusting a forecast has positive expected reward only when the forecast is right with probability above the target accuracy q^* . As its forecasts improve, the policy therefore drops the corpus’s abstention before its accuracy reaches the target.

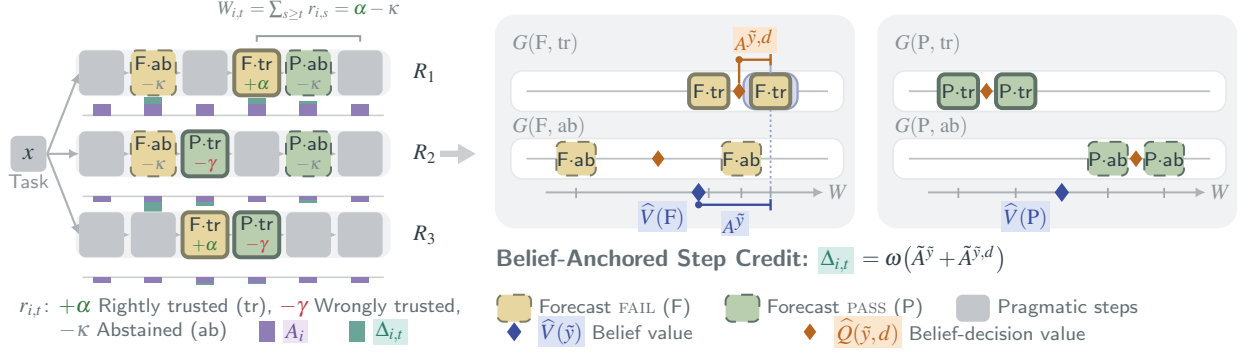


Figure 3 BAPO credit. Left: three rollouts of one task; a conclusive step shows its forecast (F or P), its decision (tr or ab), and its decision reward $r_{i,t}$; a bracket marks one decision return $W_{i,t}$, and the bar under every step is its credit $A_{i,t} = A_i + \Delta_{i,t}$ (Eq. 6); pragmatic steps carry A_i alone. Right: each white strip is a belief-decision group $G(\tilde{y}, d)$ inside its belief group $G(\tilde{y})$; for the ringed step, the gap to the belief value is its belief-relative advantage and the gap to the belief-decision value its decision-relative advantage.

Belief-anchored advantages. Crediting a step on its own needs an anchor to compare it against, and the existing approaches (Wang et al., 2026; Feng et al., 2025) are not readily applicable in our setting. BAPO instead uses the policy’s predicted outcome as a recurring grouping key, comparing steps with the same forecast across rollouts of the same task, regardless of trajectory position (Figure 3). The episode advantage is mean-only (Liu et al., 2025), $A_i := R_i - \frac{1}{N} \sum_{j=1}^N R_j$. An anchor value $\tilde{y}$ is PASS or FAIL on repository issues. We define the belief group of $\tilde{y}$ as $G(\tilde{y}) := \{(j, s) \in \mathcal{C} : \hat{y}_{j,s}^{\text{tst}} = \tilde{y}\}$, excluding the trusted crash forecasts that score zero above, its belief-decision groups as $G(\tilde{y}, d) := \{(j, s) \in G(\tilde{y}) : d_{j,s} = d\}$ for $d \in \{\text{TRUST}, \text{ABSTAIN}\}$, and the anchored steps as $\mathcal{G} := G(\text{PASS}) \cup G(\text{FAIL})$. The key is the predicted check field on repository issues and the predicted exit field on the terminal harness, where the check is evaluated once per rollout. The resulting baselines are conditional on the belief: trusting a predicted FAIL and trusting a predicted PASS have different values, and a single global baseline would mix them. For an anchored step $(i, t) \in \mathcal{G}$ with predicted key field $\tilde{y}$ and decision $d := d_{i,t}$, the *belief value*, the *belief-decision value*, and the *belief-relative* and *decision-relative advantages* are

$$\begin{aligned} \hat{V}(\tilde{y}) &:= \mathbb{E}_{G(\tilde{y})}[W], & A_{i,t}^{\tilde{y}} &:= W_{i,t} - \hat{V}(\tilde{y}), \\ \hat{Q}(\tilde{y}, d) &:= \mathbb{E}_{G(\tilde{y}, d)}[W], & A_{i,t}^{\tilde{y}, d} &:= W_{i,t} - \hat{Q}(\tilde{y}, d), \end{aligned} \quad (5)$$

where $\mathbb{E}_F$ is the uniform mean over a set F of steps, and $\hat{V}$ and $\hat{Q}$ play the role of a value and an action value, computed on belief groups instead of on states. The anchor and the decision are the policy’s own output. With the group values and the anchor groups held fixed, subtracting $\hat{V}$ or $\hat{Q}$ is therefore an ordinary baseline for the tokens written after them. For the reasoning tokens, each stream adds a shaping term bounded by the range of its group values times the expected score norm (Appendix C.2). The two streams are standardized within the task, with the divisor floored at the reward scale $\alpha + \gamma$; the standardized advantages are written $\tilde{A}$. A tilde on y marks a group key, and a tilde on A marks standardization. The combined step term is clipped, a degenerate group falls back to the task mean, and a task with no trusted conclusive step gets no step term. Before that clipping and those fallbacks, BAPO assigns

$$A_{i,t} := A_i + \Delta_{i,t}, \quad \Delta_{i,t} := \omega(\tilde{A}_{i,t}^{\tilde{y}} + \tilde{A}_{i,t}^{\tilde{y}, d}) \text{ for } (i, t) \in \mathcal{G}, \quad \Delta_{i,t} := 0 \text{ otherwise}, \quad (6)$$

with $\omega = \frac{1}{2}$. Before standardization, the step term is the decision return minus one baseline, $\omega(A^{\tilde{y}} + A^{\tilde{y},d}) = W_{i,t} - \frac{1}{2}(\hat{V}(\tilde{y}) + \hat{Q}(\tilde{y}, d))$: the belief-decision value shrunk halfway toward the belief value, which stabilizes it. This equals $A_{i,t}^{\tilde{y},d} + \frac{1}{2}\hat{A}(\tilde{y}, d)$ with $\hat{A} := \hat{Q} - \hat{V}$ the decision advantage: the step’s deviation within its decision group counts at full weight, and the credit for the decision itself at half (Proposition 3).

RL objective. The three signals combine into one objective. For the k -th token $w_{i,t,k}$ of step t , the importance ratio $\varrho_{i,t,k}(\theta) := \pi_\theta(w_{i,t,k} \mid h_{i,t}, w_{i,t,<k}) / \pi_{\text{old}}(w_{i,t,k} \mid h_{i,t}, w_{i,t,<k})$ compares the current policy with π_{old} , the policy that generated the batch. With the expectation over $x \sim \mathcal{D}$ and $\tau_{1:N} \sim \pi_{\text{old}}(\cdot \mid x)$, the training objective is

$$\begin{aligned} \mathcal{L}(\theta) &= -\mathbb{E}_{x, \tau_{1:N}} \left[\frac{1}{Z} \sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{k \in \mathcal{K}_{i,t}} \ell_{i,t,k}(\theta) \right] + \eta \hat{D}_{\text{KL}}(\pi_\theta \parallel \pi_{\text{ref}}) + \lambda \mathcal{L}_{\text{pred}}(\theta), \\ \ell_{i,t,k}(\theta) &:= \min \left(\varrho_{i,t,k} A_{i,t}, \text{clip}(\varrho_{i,t,k}, 1-\epsilon_{\text{lo}}, 1+\epsilon_{\text{hi}}) A_{i,t} \right). \end{aligned} \quad (7)$$

Here η and λ weight the KL and prediction terms, $(\epsilon_{\text{lo}}, \epsilon_{\text{hi}})$ is the asymmetric clip (Yu et al., 2025a), and $\hat{D}_{\text{KL}}$ is the k_3 estimator (Schulman, 2020) anchored at the frozen warm start π_{ref} . Reasoning tokens receive gradient from both terms. The policy term weights them by $A_{i,t}$, and $\mathcal{L}_{\text{pred}}$ back-propagates through them as part of $c_{i,t}$. Through these shared parameters, the prediction loss shapes the representation that the action is conditioned on (Gangwani et al., 2020). Algorithm 2 gives the full pipeline; notation, outcome channels, implementation rules, and prompts are in Appendices B, E, F, and O.

3.4 ANTE-TTO: Action Selection by the World-Model Prediction

The trained foresight can also be used at inference. The forecast and the decision already tell which candidate the policy expects to succeed, so test-time foresight needs neither the extra executions of best-of- n (Cobbe et al., 2021) nor a second model. When the policy abstains or predicts FAIL, ANTE-TTO samples K alternative messages and ranks them together with the original by their own beliefs. The score is $s(\beta) = \phi(\hat{y}^{\text{tst}}) + \mathbf{1}[\hat{y}^{\text{exe}} = 1]$ with $\phi = 3, 1, 0$ for PASS, NA, and FAIL. It executes the highest-ranked candidate, keeps the original on ties, and changes no weights (Algorithm 3). Its regret follows from Proposition 2 with $K + 1$ candidates, plus the spread of a point forecast at uncertain steps (Corollary 3).

4 Experiments

4.1 Experimental Setup

Model and Environment. We train and evaluate ANTE on two agent harnesses: repository issues under R2E-Gym (Jain et al., 2025) at 7B, 14B, and 32B, and terminal tasks at 9B with the Tmax agent (Iverson et al., 2026). The 7B and 14B models build on Qwen2.5-Coder-Instruct (Hui et al., 2024), the 32B model on Qwen3-32B (Yang et al., 2025a), and the 9B model on Qwen3.5-9B (Qwen Team, 2026). The belief schema and the advantage are the same on both harnesses; the outcome space, the anchor key, and the decision reward are set per harness (§3.1, Appendix I).

Data and Comparisons. SFT uses paired corpora containing 12,500 verified successful trajectories over 6,729 repository tasks and 30,161 over 10,823 terminal tasks from the tmax-15k pool (Iverson et al., 2026). We compare ANTE with two controlled baselines: *Reactive*, which follows a conventional agent-training paradigm in which the policy interleaves thinking and tool calls; and *Retrospective*, which augments *Reactive* with an explicit memory state summarizing the trajectory so far. Neither includes ANTE’s outcome forecasts or trust/abstain decisions. Their paired SFT corpora share source trajectories and tool calls. Each variant then generates fresh on-policy rollouts from its own SFT checkpoint using matched RL pools of 6,566 repository tasks and 3,558 terminal tasks. To ablate BAPO, we additionally evaluate ANTE (w/o BAPO), which retains the belief state and prediction loss but uses episode-level policy advantages without decision rewards or belief-anchored step adjustments.

Evaluation. We evaluate on SWE-bench Verified (Jimenez et al., 2024), on three terminal benchmarks (TB-Dev, TB-Lite (OpenThoughts-Agent team et al., 2026), and Terminal-Bench 2.0 (Merrill et al., 2026)), and

Table 1 ANTE outperforms both baselines at every scale on SWE-bench Verified (pass@1, %). $-$: without protocol annotated (Appendix O); $\uparrow$: gain over the variant’s SFT checkpoint, and for ANTE-TTO over ANTE; $\pm$: s.e. over 3 evaluation runs of one checkpoint, for the rows run three times.

	Method	7B	14B	32B
SFT	R2E-Gym Agent	19.0	26.8	25.0
	Reactive	23.4	40.4	45.8
	Retrospective	23.0 $^-$ / 22.4	41.0 $^-$ / 40.2	47.2 $^-$ / 48.2
	ANTE	22.2 $^-$ / 24.8	40.2 $^-$ / 41.4	48.8 $^-$ / 46.4
RL	Reactive-RL	24.4 $\uparrow^{1.0} \pm 0.1$	41.8 $\uparrow^{1.4} \pm 0.4$	51.0 $\uparrow^{5.2} \pm 0.6$
	Retrospective-RL	25.2 $\uparrow^{2.8} \pm 0.3$	43.4 $\uparrow^{3.2} \pm 0.3$	52.4 $\uparrow^{4.2} \pm 0.4$
	ANTE (w/o BAPO)	25.4 $\uparrow^{0.6}$	43.6 $\uparrow^{2.2}$	54.2 $\uparrow^{7.8}$
	ANTE	27.2 $\uparrow^{2.4} \pm 1.7$	46.2 $\uparrow^{4.8} \pm 0.9$	57.2 $\uparrow^{10.8} \pm 1.1$
	Δ Reactive baseline	+2.8	+4.4	+6.2
	+ ANTE-TTO	28.6 $\uparrow^{1.4}$	48.6 $\uparrow^{2.4}$	57.8 $\uparrow^{0.6}$

R2E-Gym Agent: at 7B and 14B, the agents released by R2E-Gym (fine-tuned Qwen2.5-Coder); at 32B, the Qwen3-32B backbone on the R2E-Gym scaffold.

Table 2 ANTE outperforms both baselines on every terminal benchmark (resolved, %). All variants build on Qwen3.5-9B and run with the Tmax agent (Iverson et al., 2026) on Harbor (Harbor Framework Team, 2025), with the matched setup of Table 1 (protocol in Appendix K). The terminal SFT checkpoints, which install the belief protocol before RL, are not evaluated separately, so $\uparrow$ is the gain over the base model rather than over each variant’s SFT checkpoint; $\pm$: s.e. over 3 evaluation runs of one checkpoint.

	TB-Dev	TB-Lite	TB-2.0
Qwen3.5-9B (base)	32.9 ± 0.8	32 ± 2.0	16.9 ± 0.7
Reactive-RL	35.6 $\uparrow^{2.7} \pm 1.6$	35 $\uparrow^3 \pm 0.9$	18.0 $\uparrow^{1.1} \pm 2.6$
Retrospective-RL	37.0 $\uparrow^{4.1} \pm 0.9$	43 $\uparrow^{11} \pm 2.1$	20.2 $\uparrow^{3.3} \pm 1.1$
ANTE (w/o BAPO)	41.1 $\uparrow^{8.2} \pm 1.2$	39 $\uparrow^7 \pm 2.2$	22.5 $\uparrow^{5.6} \pm 0.4$
ANTE	42.5 $\uparrow^{9.6} \pm 3.3$	45 $\uparrow^{13} \pm 1.3$	24.7 $\uparrow^{7.8} \pm 1.3$
Δ Reactive baseline	+6.9	+10.0	+6.7

on eleven out-of-distribution reasoning benchmarks. The RL baselines train the same corrected group-relative objective in the same setting, with the belief components removed. Each starts from the SFT checkpoint of its own corpus variant, so every matched pair isolates the belief. ANTE-TTO also runs one trajectory per task: at each trigger step it samples $K = 3$ alternatives at temperature 1.0, scores them by their beliefs, and executes only the winner (§3.4, Appendix G). All other entries are single greedy attempts (temperature 0). Implementation details and run-to-run variation are in Appendices F, J, and K.

4.2 Main Results

Our main results are shown in Tables 1 and 2. Adding the belief state and training with BAPO improves over both baselines at every scale and on every benchmark. Concretely, on SWE-bench Verified, ANTE leads both baselines at every scale, by 2.8, 4.4, and 6.2 points over Reactive and by 2.0, 2.8, and 4.8 over Retrospective at 7B, 14B, and 32B, and reaches 57.2% pass@1 at 32B (Table 1). On the terminal tasks, it leads both baselines on all three benchmarks, by 6.7 to 10.0 points over Reactive and by 2.0 to 5.5 over Retrospective, and resolves 24.7% of Terminal-Bench 2.0 (Table 2).

Besides, we find that without BAPO, the belief state alone already improves over Reactive, by 1.0 to 3.2 points on SWE-bench Verified and 4.0 to 5.5 on the terminal tasks, and matches or exceeds Retrospective except on TB-Lite. Furthermore, training with BAPO adds 1.8, 2.6, and 3.0 points on top of the belief state at 7B, 14B, and 32B, about half or more of ANTE’s lead over Reactive, and 1.4 to 6.0 points on the terminal tasks, so a large part of the gain comes from BAPO’s step-level credit (§4.5). Furthermore, our experiments corroborate the theoretical analysis: declining SFT loss supports belief-augmented initialization (Proposition 1; Figure 12), while improving forecast accuracy and consistent ANTE-TTO gains show that learned beliefs are both predictive and actionable (Proposition 2; Figure 5; Table 1).

We also compare ANTE with published open-weight agents. This comparison asks whether the resulting agents are competitive with existing systems in the literature at comparable model scales. On SWE-bench Verified, ANTE leads the task-trained agents at every scale; above 30B it is ahead of SWE-World, FrogBoss (Sonwane et al., 2025), and CWM, and of every 72B method (Table 9, Appendix J). On Terminal-Bench 2.0, ANTE-9B leads every terminal agent below 20B that we compare with, including ECHO, which predicts observations but never commits to a scored forecast (Table 10, Appendix K).

4.3 Out-of-Domain Generalization

Tables 1 and 2 measure ANTE inside the harnesses it was trained on; we next ask whether the foresight it learns there transfers beyond agentic coding. Table 3 shows ANTE-32B and Reactive-RL on eleven benchmarks spanning code execution prediction (Gu et al., 2024; Jain et al., 2024), state tracking (BIG-Bench Hard, BBH) (Suzgun et al., 2022), mathematics (Hendrycks et al., 2021; Lightman et al., 2023; math-ai, 2025a,b; Lewkowycz et al., 2022), and science (Wang et al., 2024b, 2023), answered with a plain chain-of-thought prompt (Wei et al., 2022). The two methods share the backbone, trajectories, and training procedure and differ only in the belief and its signals, and therefore we can study what the belief changes in general reasoning. ANTE leads on all datasets by 6.2 points on average and most on AIME’25 and MMLU-Pro, and the traces in Appendix N.5 show the foresight also works outside the harness.

Table 3 Out-of-distribution generalization of ANTE-32B.

Benchmark	Reactive	ANTE	Δ
Execution Prediction			
CruxEval-O	83.6	85.4	$\uparrow 1.8$
CruxEval-I	72.2	78.8	$\uparrow 6.6$
LCB-exec	91.4	94.2	$\uparrow 2.8$
State Tracking			
BBH-hard	87.2	90.9	$\uparrow 3.7$
BBH-state	88.6	91.7	$\uparrow 3.1$
Math Reasoning			
MATH500	94.8	97.2	$\uparrow 2.4$
AIME’25	63.3	80.0	$\uparrow 16.7$
AMC23	90.0	97.5	$\uparrow 7.5$
Minerva	38.6	46.0	$\uparrow 7.4$
Scientific Reasoning			
MMLU-Pro	72.8	83.2	$\uparrow 10.4$
SciBench	66.0	71.4	$\uparrow 5.4$
Mean of 11	77.1	83.3	$\uparrow 6.2$

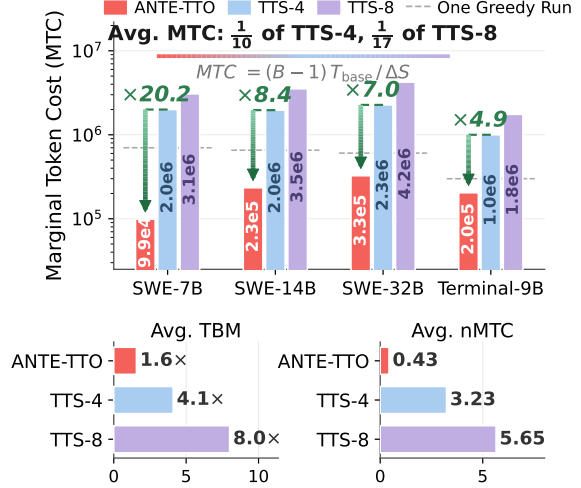


Figure 4 ANTE-TTO cost analysis. B_{tok} (TBM in the panels): inference tokens per task relative to one greedy attempt; $\text{nMTC} = (B_{\text{tok}} - 1) / \Delta S$: greedy attempts spent per point of score gain; axes show inference tokens as 10^x .

4.4 ANTE-TTO: Gains and Overhead of Test-Time Foresight

We evaluate ANTE-TTO as a test-time scaling strategy, measuring its gains in task success alongside the additional inference cost. ANTE-TTO improves task success at every scale (Table 1) while resampling only at trigger steps, scoring the candidates by the beliefs the policy already emits, and executing exactly one (§3.4). Best-of- n sampling (TTS- n), execution-based reranking (Chen et al., 2022), and process-reward reranking (Lightman et al., 2023) instead multiply full rollouts, execute every candidate, or query an external scorer. Measured from call counts, ANTE-TTO runs at $B_{\text{tok}} = 1.31\times$ at 7B against $4.42\times$ for TTS-4, and at $1.6\times$ against $4.1\times$ averaged over its four settings (Figure 4); Figure 14 shows ANTE-TTO case studies on different harnesses end to end.

4.5 Why ANTE Works

The world model sharpens and the trust decision is learned. To understand the behavior that yields these results, we analyze ANTE’s trust decisions, the relationship between its beliefs and task outcomes, and representative trajectories. Figure 5 traces the belief through RL. The hit rate of conclusive predictions rises at every scale (a), from 44% to 69% at 32B, so the forecasts the policy acts on become more accurate as training proceeds. The trust rate doubles (b), the accuracy of trusted predictions rises past q^* , the level above which trusting is rewarded (c), and abstention concentrates on the steps that remain uncertain (d). The belief also outperforms SWE-World’s dedicated 32B reward model (Sun et al., 2026) as a judge of its trajectories (Table 11). Together with the ANTE-TTO gains of Table 1, this is the empirical side of Proposition 2. RL

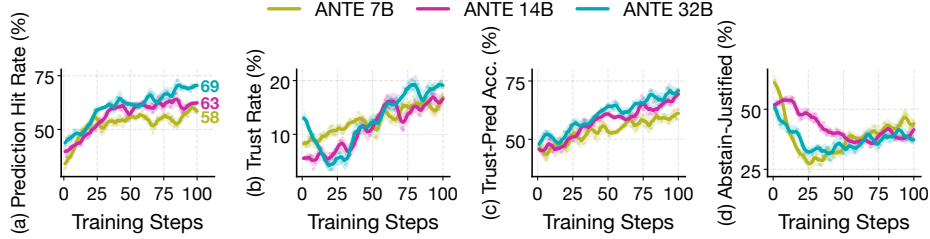


Figure 5 The world model sharpens and the trust decision is learned. (a) The prediction hit rate of ANTE’s world model on conclusive steps rises throughout RL at every scale. (b)–(d) BAPO analysis: (b) the trust rate doubles, (c) the accuracy of trusted predictions rises, and (d) abstention concentrates on the steps that remain uncertain.

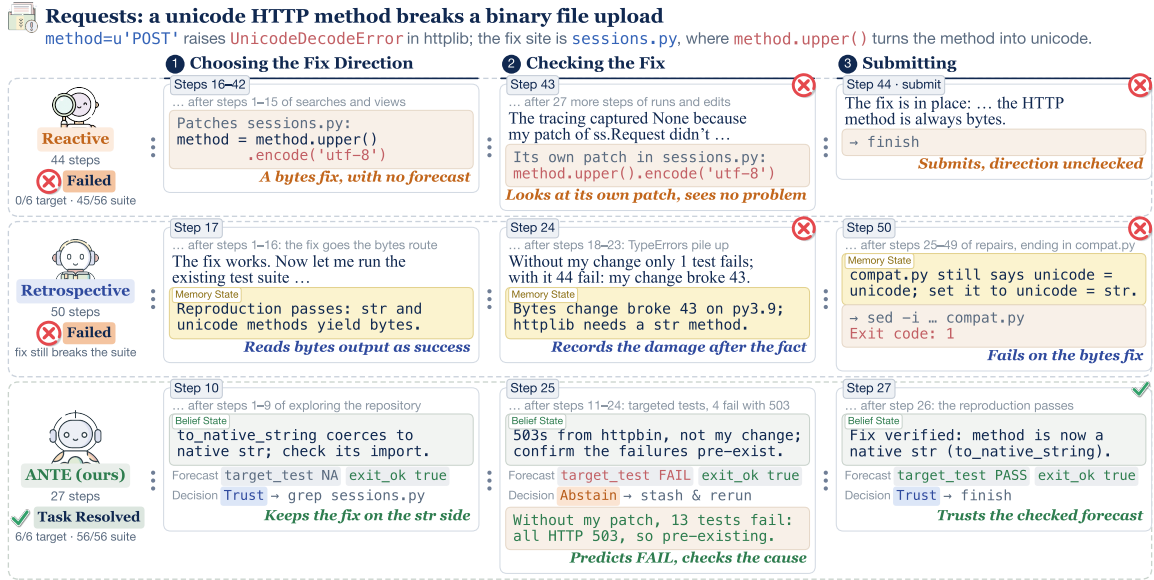


Figure 6 Case study. The internal belief helps reinforce ANTE’s world model reasoning for coding agents (task: instance `psf__requests-1724` of SWE-bench Verified).

lowers the forecast error that bounds the selection regret, the forecasts carry outcome signal, and selecting by them raises success. The foresight is therefore not only predictive but also actionable.

Case study. Figure 6 shows how the three models handle a repository issue. In this case, uploading a file with the method `u'POST'`, a Unicode string, raises a `UnicodeDecodeError`. The fix is to convert the method to the native string type, which is bytes on Python 2 and str on Python 3. Reactive reasons only about the immediate error and encodes the method to bytes. It is not aware of the environment the fix must run in, so when Python 3 rejects the bytes method, it dismisses this as outside the issue and submits the fix. For Retrospective, the memory records only past steps and indicates nothing about what its bytes edit will do. ANTE’s belief predicts what the next edit will change. It scopes the bug to Python 2 (“force bytes in py2”), so it uses the existing native-string helper instead of forcing bytes everywhere. This foresight ability makes ANTE resolve tasks more robustly (full trace in Appendix N).

5 Conclusion

With ANTE, we want coding agents to foresee the consequences of their actions and to let real execution correct their beliefs. During training, ANTE’s world model forms an internal belief within the policy: what the agent has established, what the code does now, and what the next command will change. The foresight ability lets ANTE improve across harnesses and model scales, and generalize to tasks in other domains. In summary, we envision agents whose beliefs span their environment’s past, present, and future, acting on what they anticipate and learning from what execution reveals.

References

- Eitan Altman. *Constrained Markov Decision Processes*. Chapman & Hall/CRC, 1999.
- Karl Johan Åström. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965.
- Ibragim Badertdinov et al. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *arXiv preprint arXiv:2505.20411*, 2025.
- Jiaxin Bai, Yue Guo, Yifei Dong, Jiaxuan Xiong, Tianshi Zheng, et al. Patchworld: Gradient-free optimization of executable world models for agent environments. *arXiv preprint arXiv:2605.30880*, 2026.
- David Barber and Felix Agakov. The IM algorithm: A variational approach to information maximization. In *Advances in Neural Information Processing Systems*, 2003.
- Shiyi Cao, Dacheng Li, Fangzhou Zhao, et al. SkyRL-Agent: Efficient RL training for multi-turn LLM agent. *arXiv preprint arXiv:2511.16108*, 2025.
- Bei Chen et al. CodeT: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*, 2022.
- Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*, 2025.
- Karl Cobbe et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- Ganqu Cui, Lifan Yuan, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.
- Hen Davidov, Nachshon Cohen, Oren Kalinsky, Yaron Fairstein, Guy Kushilevitz, Ram Yazdi, and Patrick Rebeschini. Knowing when to quit: A principled framework for dynamic abstention in LLM reasoning. In *International Conference on Machine Learning*, 2026.
- DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.
- Yifeng Ding, Hung Le, Songyang Han, Kangrui Ruan, Zhenghui Jin, Varun Kumar, Zijian Wang, and Anoop Deoras. Empowering multi-turn tool-integrated agentic reasoning with group turn policy optimization. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 42409–42423, 2026.
- 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.
- Kanishk Gandhi, Shivam Garg, et al. Endless terminals: Scaling RL environments for terminal agents. *arXiv preprint arXiv:2601.16443*, 2026.
- Tanmay Gangwani, Joel Lehman, Qiang Liu, and Jian Peng. Learning belief representations for imitation learning in POMDPs. In *Conference on Uncertainty in Artificial Intelligence*, 2020.
- Alex Gu et al. CRUXEval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024.
- Harbor Framework Team. Harbor: A framework for evaluating and improving agents. <https://github.com/harbor-framework/harbor>, 2025.
- Dan Hendrycks et al. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021.
- Binyuan Hui et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.
- Hamish Ivison et al. Tmax: A simple recipe for terminal agents. *arXiv preprint arXiv:2606.23321*, 2026.
- Naman Jain et al. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.
- Naman Jain et al. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *arXiv preprint arXiv:2504.07164*, 2025.

- Carlos E. Jimenez et al. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.
- Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.
- Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2002.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023.
- Aitor Lewkowycz et al. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*, 2022.
- Shuquan Lian, Juncheng Liu, Yazhe Chen, Yuhong Chen, and Hui Li. SWE-AGILE: A software agent framework for efficiently managing dynamic reasoning context. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.
- Hunter Lightman et al. Let’s verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.
- Zichen Liu et al. Understanding R1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.
- Ning Lu, Baijiong Lin, Shengcai Liu, Jiahao Wu, Haoze Lv, Yanbin Wei, Lingting Zhu, Shengju Qian, Xin Wang, Ying-Cong Chen, Qi Wang, and Ke Tang. Policy and world modeling co-training for language agents. *arXiv preprint arXiv:2606.02388*, 2026.
- Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. DeepSWE: Training a fully open-sourced, state-of-the-art coding agent by scaling RL. <https://www.together.ai/blog/deepswe>, 2025. Agentica and Together AI blog post, July 2025.
- Yingwei Ma et al. Lingma SWE-GPT: An open development-process-centric language model for automated software improvement. *arXiv preprint arXiv:2411.00622*, 2024.
- math-ai. AIME 2025. <https://huggingface.co/datasets/math-ai/aime25>, 2025a.
- math-ai. AMC 2023. <https://huggingface.co/datasets/math-ai/amc23>, 2025b.
- Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.
- Meta FAIR CodeGen Team. CWM: An open-weights LLM for research on code generation with world models. *arXiv preprint arXiv:2510.02387*, 2025.
- Ansong Ni, Miltiadis Allamanis, Arman Cohan, Yinlin Deng, Kensen Shi, Charles Sutton, and Pengcheng Yin. NExT: Teaching large language models to reason about code execution. In *International Conference on Machine Learning*, 2024.
- NovaSky Team. SkyRL-v0: Train real-world long-horizon agents via reinforcement learning. Model release and blog, NovaSky-AI, <https://novasky-ai.github.io/posts/skyrl-v0/>, 2025.
- Open Thoughts Team. OpenThinker-Agent-v1. Model release, <https://huggingface.co/open-thoughts/OpenThinker-Agent-v1>, 2025.
- OpenHands Team. Introducing OpenHands LM 32B – a strong, open coding agent model. All Hands AI blog, <https://www.openhands.dev/blog/introducing-openhands-lm-32b---a-strong-open-coding-agent-model>, 2025.
- OpenThoughts-Agent team. OpenThoughts-TB-Dev. <https://huggingface.co/datasets/open-thoughts/OpenThoughts-TB-dev>, 2025.
- OpenThoughts-Agent team, Snorkel AI, and Bespoke Labs. OpenThoughts-TB Lite: A high-signal benchmark for iterating on terminal agents. <https://www.openthoughts.ai/blog/openthoughts-tblite>, 2026.
- Jiayi Pan et al. Training software engineering agents and verifiers with SWE-Gym. In *International Conference on Machine Learning*, 2025.
- Renjie Pi, Grace Lam, Mohammad Shoeybi, Pooya Jannaty, Bryan Catanzaro, and Wei Ping. On data engineering for scaling llm terminal capabilities. *arXiv preprint arXiv:2602.21193*, 2026.

- Qwen Team. Qwen3-Coder-480B-A35B-Instruct model card. <https://huggingface.co/Qwen/Qwen3-Coder-480B-A35B-Instruct>, 2025.
- Qwen Team. Qwen3.5-9B model card. <https://huggingface.co/Qwen/Qwen3.5-9B>, 2026.
- Kangrui Ruan, Junzhe Zhang, Xuan Di, and Elias Bareinboim. Causal imitation for markov decision processes: A partial identification approach. *Advances in neural information processing systems*, 37:87592–87620, 2024.
- John Schulman. Approximating KL divergence. <http://joschu.net/blog/kl-approx.html>, 2020.
- John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys)*, 2025.
- Vaishnavi Shrivastava, Piero Kauffmann, Ahmed Awadallah, and Dimitris Papailiopoulos. ECHO: Terminal agents learn world models for free. *arXiv preprint arXiv:2605.24517*, 2026.
- Joykirat Singh, Zaid Khan, Archiki Prasad, Justin Chih-Yao Chen, Akshay Nambi, Hyunji Lee, Elias Stengel-Eskin, and Mohit Bansal. Agent-BRACE: Decoupling beliefs from actions in long-horizon tasks via verbalized state uncertainty. *arXiv preprint arXiv:2605.11436*, 2026.
- Huatong Song, Lisheng Huang, Shuang Sun, Jinhao Jiang, Ran Le, Daixuan Cheng, Guoxin Chen, Yiwen Hu, Zongchao Chen, Yiming Jia, Wayne Xin Zhao, Yang Song, Tao Zhang, and Ji-Rong Wen. SWE-Master: Unleashing the potential of software engineering agents via post-training. *arXiv preprint arXiv:2602.03411*, 2026.
- Atharv Sonwane et al. BugPilot: Complex bug generation for efficient learning of SWE skills. *arXiv preprint arXiv:2510.19898*, 2025.
- Shuang Sun, Huatong Song, Lisheng Huang, Jinhao Jiang, Ran Le, Zhihao Lv, Zongchao Chen, Yiwen Hu, Wenyang Luo, Wayne Xin Zhao, Yang Song, Hongteng Xu, Tao Zhang, and Ji-Rong Wen. SWE-World: Building software engineering agents in docker-free environments. *arXiv preprint arXiv:2602.03419*, 2026.
- Mirac Suzgun et al. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.
- Lingxiao Tang, He Ye, Zhaoyang Chu, Muyang Ye, et al. Execverify: White-box RL with verifiable stepwise rewards for code execution reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026a.
- Wenjie Tang, Minne Li, Sijie Huang, Liquan Xiao, and Yuan Zhou. Rewarding beliefs, not actions: Consistency-guided credit assignment for long-horizon agents. *arXiv preprint arXiv:2605.20061*, 2026b.
- Chaofan Tao, Jierun Chen, Yuxin Jiang, et al. SWE-Lego: Pushing the limits of supervised fine-tuning for software issue resolving. *arXiv preprint arXiv:2601.01426*, 2026.
- Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization. In *International Conference on Learning Representations*, 2019.
- The Terminal-Bench Team. Terminal-Bench 1.0: A benchmark for LLMs on complicated tasks in the terminal. <https://github.com/harbor-framework/terminal-bench-1>, 2025.
- Hao Wang, Rui Li, Lei Sha, and Jie M. Zhang. StepCodeReasoner: Aligning code reasoning with stepwise execution traces via reinforcement learning. *arXiv preprint arXiv:2605.11922*, 2026.
- Haoran Wang et al. SWE-Dev: Building software engineering agents with training and inference scaling. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025a.
- Kangrui Wang et al. VAGEN: Reinforcing world model reasoning for multi-turn VLM agents. In *Advances in Neural Information Processing Systems*, 2025b.

- Qi Wang, Hongzhi Zhang, Jia Fu, Kai Fu, Yahui Liu, Tinghai Zhang, Chenxi Sun, Gangwei Jiang, Jingyi Tang, Xingguang Ji, Yang Yue, Jingyuan Zhang, Fuzheng Zhang, Kun Gai, and Guorui Zhou. Klear-AgentForge: Forging agentic intelligence through posttraining scaling. *arXiv preprint arXiv:2511.05951*, 2025c.
- Xiaoxuan Wang et al. SciBench: Evaluating college-level scientific problem-solving abilities of large language models. *arXiv preprint arXiv:2307.10635*, 2023.
- Xingyao Wang et al. OpenHands: An open platform for AI software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024a.
- Yubo Wang et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024b.
- Jason Wei et al. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.
- Sean Wu, Fredrik K. Gustafsson, Edward Phillips, Boyan Gao, Anshul Thakur, and David A. Clifton. BAS: A decision-theoretic approach to evaluating large language model confidence. *arXiv preprint arXiv:2604.03216*, 2026a.
- Siwei Wu, Yizhi Li, Yuyang Song, Wei Zhang, Yang Wang, Riza Batista-Navarro, Xian Yang, Mingjie Tang, Bryan Dai, Jian Yang, et al. Large-scale terminal agentic trajectory generation from dockerized environments. *arXiv preprint arXiv:2602.01244*, 2026b.
- Chengxing Xie et al. SWE-Fixer: Training open-source LLMs for effective and efficient GitHub issue resolution. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025.
- An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.
- John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, et al. SWE-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems*, 2025b.
- John Yang et al. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.
- Zonghan Yang et al. Kimi-Dev: Agentless training as skill prior for SWE-agents. *arXiv preprint arXiv:2509.23045*, 2025c.
- Shunyu Yao et al. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.
- Qiyang Yu et al. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, 2025a.
- Xiao Yu, Baolin Peng, Michel Galley, Hao Cheng, Qianhui Wu, Janardhan Kulkarni, Suman Nath, Zhou Yu, and Jianfeng Gao. Dyna-Mind: Learning to simulate from experience for better AI agents. *arXiv preprint arXiv:2510.09577*, 2025b.
- Xiao Yu et al. Reinforcement world model learning for LLM-based agents. *arXiv preprint arXiv:2602.05842*, 2026.
- Liang Zeng et al. Skywork-SWE: Unveiling data scaling laws for software engineering in LLMs. *arXiv preprint arXiv:2506.19290*, 2025.
- Hao Zhang, Mingjie Liu, Shaokun Zhang, et al. ProRL Agent: Rollout-as-a-service for RL training of multi-turn LLM agents. *arXiv preprint arXiv:2603.18815*, 2026.
- Deyu Zou, Yongqiang Chen, Jianxiang Wang, Haochen Yang, Mufei Li, Qing Da, James Cheng, Pan Li, and Yu Gong. Reducing belief deviation in reinforcement learning for active reasoning of LLM agents. In *International Conference on Learning Representations*, 2026.

Appendix

Appendix Contents

A Preliminaries	16
B Notation	17
C Theoretical analysis of the ANTE objective	17
C.1 Setting and the quantities the analysis controls	18
C.2 Decomposition of the anchored advantage	18
C.3 Belief-augmented imitation	20
C.4 Task reward under a world-model accuracy budget	20
C.5 Information-theoretic guarantees of the prediction loss	21
D Corpus curation	24
D.1 Task screening and trajectory collection	24
D.2 Execution-derived labels and checks	25
D.3 Paired corpus construction and protocol-leakage filtering	25
D.4 Corpus statistics and evaluation task lists	26
E Belief state specification and outcome evaluation	26
E.1 Fields of the belief state	26
E.2 Outcome channels, parsing rules, and decision reward per harness	27
F Training setup and infrastructure	28
F.1 Supervised fine-tuning records	28
F.2 The optimization objective	28
F.3 Break-even derivation of the reward coefficients	29
F.4 Compute and software stack	30
G ANTE-TTO: candidate score, algorithm, and examples	30
G.1 Scoring rule and inference algorithm	30
G.2 Examples on different harnesses	31
H A worked example of BAPO	31
I Terminal instantiation and configuration differences	32
J Evaluation against concurrent methods on SWE-bench Verified	34
K Terminal evaluation protocol	35
L Additional analyses of the belief under RL	36
M Failure modes without foresight	36
N Case studies	37
N.1 Foresight guides test-time optimization	37
N.2 Memory extends context but not foresight	39
N.3 The belief sharpens as the world model trains	39
N.4 A confident forecast of failure is still a commitment	40
N.5 Foresight transfers beyond agentic coding	40
O Instruction templates	40
O.1 Repository templates	40
O.2 Terminal and collection templates	44

A Preliminaries

Coding agents as POMDPs. We formalize a coding agent working on a task inside its container as a partially-observable Markov decision process (POMDP) $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \Omega, \mathcal{R})$ (Åström, 1965; Kaelbling et al., 1998). The latent state s_t is the full repository and container, an action a_t is one tool call, $\mathcal{T}(s_{t+1} | s_t, a_t)$ executes it, the observation $o_{t+1} \sim \Omega(\cdot | s_{t+1}, a_t)$ is its textual output, and the reward $\mathcal{R}$ is a terminal outcome in $\{0, 1\}$. The agent therefore acts on the history h_t of everything it has written and observed. Over histories the process is Markov with kernel $P_{\text{env}}(o_{t+1} | h_t, a_t)$, so policy gradients apply unchanged. We use the corrected group-relative policy gradient of Dr.GRPO and DAPO (Shao et al., 2024; Liu et al., 2025; Yu et al., 2025a) for the policy term of Eq. 7 (Appendix F). The belief $b_t = P(s_t | h_t)$ suffices for optimal control but cannot be computed or checked for a repository, so learning agents keep a belief representation instead, a function of the history trained to predict what the latent state determines (Gangwani et al., 2020).

Coding tasks and execution-based verification. A repository issue (Jimenez et al., 2024; Jain et al., 2025) consists of a containerized repository at a fixed commit, an issue description, and a hidden test suite $\mathcal{U}$. It is resolved when the submitted patch passes every fail-to-pass and pass-to-pass test in $\mathcal{U}$. A terminal task states a goal in natural language inside an isolated container, and a hidden checker grades the final state. In both harnesses, each step is one assistant message that reasons and then makes exactly one tool call, and the environment replies only in text. Execution can check one part of the belief exactly: the deterministic verifier ψ of §3.1 (Jain et al., 2025) maps each response to an outcome in $\mathcal{Y}$, or to $\perp$ when it determines none. The realized outcome $y_t := \psi(o_{t+1}, \chi(a_t))$ therefore needs no judge, learned reward model, or simulator, and its distribution given (h_t, a_t) is fixed by P_{env} . ANTE trains the policy to predict y_t before a_t runs and scores the forecast against the realized outcome (§3). The belief state of §3.1 is the written form of this belief representation, and its forecast is the part that execution checks.

B Notation

Table 4 Notation. Hats mark estimates; $\tilde{y}$ denotes a value of the predicted key field (a group key) and $\tilde{A}$ a standardized advantage.

Symbol	Meaning	Definition or reference
<i>Environment and rollouts</i>		
$(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \Omega, \mathcal{R})$	POMDP of a harness	Appendix A
$P_{\text{env}}(o_{t+1} \mid h_t, a_t)$	Kernel of the history process	Appendix A
b_t	Posterior belief	$P(s_t \mid h_t)$
$\mathcal{Y} = \mathcal{Y}^{\text{tst}} \times \{0, 1\}$	Outcome space	$\mathcal{Y}^{\text{tst}} = \{\text{PASS}, \text{FAIL}, \text{NA}\}$; §3.1
$x \sim \mathcal{D}$; $\mathcal{D}_{\text{BT}}$	Task; BeliefTraj corpus	§3.3; §3.2
π_θ ; π_{old} ; π_{ref}	Policy; policy that generated the batch; reference policy	Eq. 7
N ; τ_i ; T_i ; $h_{i,t}$	Rollouts per task; rollout i ; its length; history before step t	§3.3
$u_{i,t} = \langle v_{i,t}, \beta_{i,t}, a_{i,t} \rangle$	Step message	Reasoning, belief state, action; §3.3
$c_{i,t} = (h_{i,t}, v_{i,t}, z_{i,t})$	Writing context of the forecast	§3.1
$\mathbb{E}_F[f]$	Mean of f over a step set F	§3.3
<i>Outcomes and belief state</i>		
χ ; ψ	Action type; execution-based verifier	$\mathcal{A} \rightarrow \{\text{EPI}, \text{PRAG}\}$; §3.1
$y_{i,t} = (y_{i,t}^{\text{tst}}, y_{i,t}^{\text{exe}})$	Realized outcome (check field, exit field)	$\psi(o_{i,t+1}, \chi(a_{i,t})) \in \mathcal{Y} \cup \{\perp\}$
$m_{i,t}$	Verifiable-step indicator	1 iff the block parses and $\psi \neq \perp$
$\mathcal{C}_i$; $\mathcal{C}$	Conclusive steps (rollout; task)	§3.1
R_i ; r_i^{fmt}	Task reward; format term	$\mathbf{1}[\text{resolved}] + r_i^{\text{fmt}}$; §3.3
A_i	Episode advantage	$R_i - \frac{1}{N} \sum_j R_j$
$\beta_{i,t} = (z_{i,t}, \hat{y}_{i,t}, d_{i,t})$	Belief state	Summary, forecast, decision; §3.1
$\hat{y}_{i,t} = (\hat{y}_{i,t}^{\text{tst}}, \hat{y}_{i,t}^{\text{exe}})$	Forecast (check field, exit field)	$\in \mathcal{Y}$
$d_{i,t}$	Decision	$\{\text{TRUST}, \text{ABSTAIN}\}$; corpus labels in Appendix D
<i>Decision reward and credit</i>		
σ ; $r_{i,t}$	Scoring rule; decision reward	$r_{i,t} = \sigma(d_{i,t}, \hat{y}_{i,t}, y_{i,t})$ on $\mathcal{C}_i$; Eq. 19
α , γ , κ	Bonus, penalty, abstention cost	Eq. 19
q^* ; q°	Target accuracy; indifference point	$\gamma/(\alpha + \gamma)$; $q^* = \kappa/(\alpha + \gamma)$; clean-exit forecasts
$W_{i,t}$	Decision return	$\sum_{s \geq t} r_{i,s}$
$G(\tilde{y})$; $G(\tilde{y}, d)$	Belief group; belief-decision group	§3.3
$\mathcal{G}$	Anchored steps	Union of the belief groups (key: predicted check field; exit field on terminal tasks)
$\hat{V}(\tilde{y})$; $\hat{Q}(\tilde{y}, d)$	Belief value; belief-decision value	Means of W over $G(\tilde{y})$, $G(\tilde{y}, d)$; Eq. 5
$A_{i,t}^{\tilde{y}}$; $A_{i,t}^{\tilde{y},d}$	Belief-relative; decision-relative advantage	$W_{i,t} - \hat{V}(\tilde{y})$; $W_{i,t} - \hat{Q}(\tilde{y}, d)$
$\hat{A}(\tilde{y}, d)$	Decision advantage	$\hat{Q}(\tilde{y}, d) - \hat{V}(\tilde{y})$
$\tilde{A}_{i,t}^{\tilde{y}}$; $\tilde{A}_{i,t}^{\tilde{y},d}$	Standardized streams	Per-task divisor $\max(\text{std}, \alpha + \gamma)$; Appendix F
$\Delta_{i,t}$; ω	Step credit; its weight	$\omega(\tilde{A}_{i,t}^{\tilde{y}} + \tilde{A}_{i,t}^{\tilde{y},d})$ on $\mathcal{G}$, else 0; Eq. 6
$A_{i,t}$	Per-step advantage	$A_i + \Delta_{i,t}$
<i>Objective, corpus, and selection</i>		
$\mathcal{L}^{\text{pred}}$	Prediction loss	Eq. 4
$\mathcal{K}_{i,t}$; $\mathcal{V}_{i,t}$; Z	Policy tokens; value tokens; token-mean denominator	Eqs. 4, 7
$\varrho_{i,t,k}$	Importance ratio	$\pi_\theta / \pi_{\text{old}}$ per policy token
$\epsilon_{\text{lo}}, \epsilon_{\text{hi}}$; η ; λ ; η_{lr}	Clip range; KL coefficient; prediction weight; learning rate	Eq. 7; Alg. 1
$s(\beta)$; $\phi(\cdot)$	Candidate score; check-field rank	Eq. 23
K	Alternatives per trigger	$K + 1$ candidates with the draft; Alg. 3
<i>Analysis (Appendix C)</i>		
$P_E, J, T, H_E, \varepsilon_{\text{SFT}}, \Lambda$; p_k	Symbols of Propositions 1 and 2	Defined in their statements
q_k, δ_k, u		
$p(\cdot \mid c_t)$; $q_\theta(\cdot \mid c_t)$	Outcome law; policy predictive over $\mathcal{Y}$	Appendix C.1, under $m_t = 1$
$\delta(c_t)$; $\mathcal{L}_t$	Forecast divergence; expected prediction loss	$\text{KL}(p \parallel q_\theta)$; $\mathbb{E}[-\log q_\theta(y_t \mid c_t)]$; Eq. 8

C Theoretical analysis of the ANTE objective

- (i) The belief-anchored advantage splits credit for a decision from credit for a step (Proposition 3, Appendix C.2).
- (ii) Read as imitation, Stage I lowers an upper bound on the gap to the demonstration process (Proposition 1).
- (iii) An idealized form of the policy term of Eq. 7 lower-bounds task performance, and the prediction weight λ indexes a monotone family of accuracy budgets (Lemma 1, Proposition 4).
- (iv) The expected prediction loss upper-bounds the uncertainty left about the checkable outcome, and $H(y_t \mid a_t) - \bar{\mathcal{L}}_t$ lower-bounds the information the writing context carries about the next latent state. The forecast divergence upper-bounds

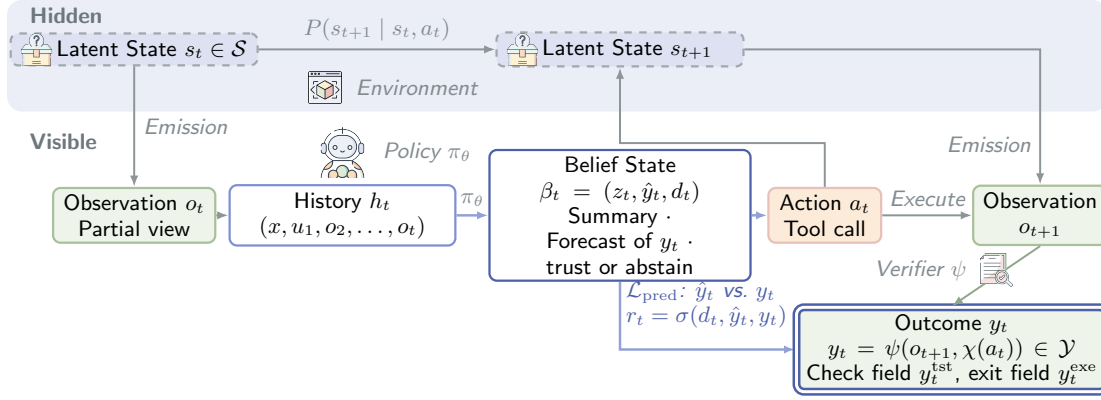


Figure 7 Belief as a point summary of the history. The forecast $\hat{y}_t$ predicts the one coordinate of the next observation that the verifier ψ reads; the prediction loss checks it against y_t , and the decision reward scores d_t against the same outcome.

the cost of acting on a forecast, in training and at test time (Proposition 5, Corollaries 1–3, Proposition 2, Appendix C.5).

C.1 Setting and the quantities the analysis controls

We work in the partially observed process of Appendix A and §3.3. A step policy $\pi(u | h)$ emits the message $u = (v, \beta, a)$ (Figure 7) given the history h , which holds the earlier messages and observations but not the current reasoning; where only the belief and the action matter we abbreviate the emission as (β, a) , the reasoning being carried inside u ; the environment kernel $P_{\text{env}}(o_{t+1} | h_t, a_t) = \sum_{s, s'} P(s_t = s | h_t) \mathcal{T}(s' | s, a_t) \Omega(o_{t+1} | s', a_t)$ depends on the step message only through the action. Every rollout ends by a horizon T (a terminated rollout emits null steps up to T with probability one under every policy considered), the terminal outcome is recorded as part of the last observation, so that $R \in [0, 1]$ is a function of the trajectory (binary in both harnesses; only boundedness is used), and the analysis uses the task outcome alone, without the bounded format term r_i^{fmt} of §3.3. P_π denotes the law of trajectories under π and the real environment, and the performance measure is $J(\pi) := \mathbb{E}_{P_\pi}[R]$. For a verifiable step ($m_t = 1$, §3.1), called a supervised step below, $p(\cdot | c_t)$ is the true conditional law of the realized outcome y_t given the writing context $c_t = (h_t, v_t, z_t)$ of §3.1, $q_\theta(\cdot | c_t)$ the policy’s predictive at the value positions renormalized over $\mathcal{Y}$, and

$$\delta(c_t) := \text{KL}(p(\cdot | c_t) \parallel q_\theta(\cdot | c_t)), \quad \bar{\mathcal{L}}_t(\theta) := \mathbb{E}[-\log q_\theta(y_t | c_t)] = H(y_t | c_t) + \mathbb{E}[\delta(c_t)], \quad (8)$$

the pointwise forecast divergence and the step- t expected prediction loss; the identity is the decomposition of cross-entropy into entropy plus divergence. The training loss $\mathcal{L}_{\text{pred}}$ of Eq. 4 is a token-normalized empirical surrogate for both. It scores the unnormalized value-token likelihood, which is at most the renormalized one, and weights supervised steps by $1/Z$. The statements below control the per-step expected prediction loss $\bar{\mathcal{L}}_t$. Quantities attached to supervised steps are taken under the rollout law conditioned on $m_t = 1$, written $\mathcal{M}$ in the prediction and information statements of Appendix C.5. The trajectory-level statements of Appendices C.3 and C.4 use the unconditioned trajectory laws. Candidates whose outcome is $\perp$ are excluded from the selection statements, and logarithms are natural. Total variation is $\text{TV}(P, Q) := \sup_E |P(E) - Q(E)|$; hence $|\mathbb{E}_P[f] - \mathbb{E}_Q[f]| \leq \text{TV}(P, Q)$ for $f \in [0, 1]$, and $\leq 2\|f\|_\infty \text{TV}(P, Q)$ in general. Pinsker’s inequality reads $\text{TV}(P, Q) \leq \sqrt{\text{KL}(P \| Q)/2}$.

C.2 Decomposition of the anchored advantage

BAPO subtracts two baselines from one decision return. The statement below shows that they are not redundant: together they split the return into the credit for choosing a decision and the credit for the step inside that decision.

Proposition 3 (Anchored-advantage decomposition). *Fix a task with rollouts $\tau_1, \dots, \tau_N$ and an anchor value $\hat{y} \in \{\text{PASS}, \text{FAIL}\}$ whose belief group $G(\hat{y})$ is non-empty, and let the belief-decision groups $G(\hat{y}, d)$,*

$d \in \{\text{TRUST}, \text{ABSTAIN}\}$, partition it, both non-empty for the final claim. Write $\mu_d := |G(\tilde{y}, d)|/|G(\tilde{y})|$ for the decision weights, and let $\hat{A}(\tilde{y}, d) := \hat{Q}(\tilde{y}, d) - \hat{V}(\tilde{y})$ denote the decision advantage of d , a quantity shared by every member of $G(\tilde{y}, d)$. Then, identically on $G(\tilde{y})$,

$$A_{i,t}^{\tilde{y}} = \hat{A}(\tilde{y}, d_{i,t}) + A_{i,t}^{\tilde{y},d}, \quad (9)$$

and under the empirical measure on $G(\tilde{y})$ the two terms are mean-zero and orthogonal,

$$\mathbb{E}_{G(\tilde{y})}[\hat{A}] = \mathbb{E}_{G(\tilde{y})}[A^{\tilde{y},d}] = 0, \quad \mathbb{E}_{G(\tilde{y})}[\hat{A} A^{\tilde{y},d}] = 0, \quad (10)$$

so that $\text{Var}_{G(\tilde{y})}(W) = \text{Var}_{G(\tilde{y})}(\hat{A}) + \text{Var}_{G(\tilde{y})}(A^{\tilde{y},d})$ splits the variance of W over the belief group into a between-decision and a within-decision part. Consequently, for the step term of the BAPO assignment before standardization, $S_{i,t} := \frac{1}{2}(A_{i,t}^{\tilde{y}} + A_{i,t}^{\tilde{y},d}) = W_{i,t} - \frac{1}{2}(\hat{V}(\tilde{y}) + \hat{Q}(\tilde{y}, d_{i,t})) = A_{i,t}^{\tilde{y},d} + \frac{1}{2}\hat{A}(\tilde{y}, d_{i,t})$ (Eq. 6 with $\omega = \frac{1}{2}$), the difference of its means over the two decisions is

$$\mathbb{E}_{G(\tilde{y}, \text{TRUST})}[S] - \mathbb{E}_{G(\tilde{y}, \text{ABSTAIN})}[S] = \frac{1}{2}(\hat{Q}(\tilde{y}, \text{TRUST}) - \hat{Q}(\tilde{y}, \text{ABSTAIN})), \quad (11)$$

so the between-decision signal is carried entirely by the decision advantage $\hat{A}$.

The identities hold before standardization and clipping. A baseline conditioned on the decision as well as the belief, $\hat{Q}$ itself, would leave the two decisions with equal group means under any fixed belief. The baseline of the BAPO step term (Eq. 6) is the average of the belief value and the belief-decision value, $\frac{1}{2}(\hat{V} + \hat{Q})$. The within-decision deviation enters once through each of the two streams, and the credit for the decision enters through the belief-relative stream alone. The average shrinks the belief-decision value toward the belief value: the belief-decision group is the sharper comparison, but with N rollouts per task a belief-decision group holds few conclusive steps, and the belief value stabilizes it. The two streams are standardized separately (Appendix F), which rescales each without changing which steps it compares.

Proof of Proposition 3. The identity follows by adding and subtracting $\hat{Q}(\tilde{y}, d_{i,t})$ in the definition $A_{i,t}^{\tilde{y}} = W_{i,t} - \hat{V}(\tilde{y})$. With μ_d the decision weights of the statement, mean-zero: $\mathbb{E}_{G(\tilde{y})}[\hat{A}] = \sum_d \mu_d (\hat{Q}(\tilde{y}, d) - \hat{V}(\tilde{y})) = 0$ since the belief-decision groups partition the belief group, and $\mathbb{E}_{G(\tilde{y})}[A^{\tilde{y},d}] = \sum_d \mu_d \mathbb{E}_{G(\tilde{y}, d)}[W - \hat{Q}(\tilde{y}, d)] = 0$. Orthogonality: $\hat{A}$ is constant on each belief-decision group, so $\mathbb{E}_{G(\tilde{y})}[\hat{A} A^{\tilde{y},d}] = \sum_d \mu_d (\hat{Q}(\tilde{y}, d) - \hat{V}(\tilde{y})) \mathbb{E}_{G(\tilde{y}, d)}[W - \hat{Q}(\tilde{y}, d)] = 0$. For the variance claim, square the identity and average over $G(\tilde{y})$; the cross term vanishes by orthogonality. For the separation claim, restricting the mean-zero property of $A^{\tilde{y},d}$ to a single belief-decision group gives $\mathbb{E}_{G(\tilde{y}, d)}[A^{\tilde{y},d}] = 0$, so the mean of S over $G(\tilde{y}, d)$ is $\frac{1}{2}\hat{A}(\tilde{y}, d) = \frac{1}{2}(\hat{Q}(\tilde{y}, d) - \hat{V}(\tilde{y}))$; subtracting the two belief-decision means cancels $\hat{V}$. $\square$

Anchor bias. The baselines $\hat{V}(\tilde{y})$ and $\hat{Q}(\tilde{y}, d)$ of Eq. 5 are indexed by the step’s own forecast and decision, so they act differently on the tokens written before and after the anchor. The reading below is the idealization in which the group values and the anchor groups are held fixed, rollouts are sampled on-policy without masks, and every forecast lies in $\{\text{PASS}, \text{FAIL}\}$. The empirical group means of Proposition 3 are their batch estimates. For the action, written after the belief, the anchor is part of the token’s context, and a baseline that is a function of the context leaves the expected score-function gradient unchanged. The decision token is baselined by $\hat{V}(\tilde{y})$ in the same way, and the difference between its two group means is carried by $\hat{A}$ (Eq. 11). For the reasoning v , written before the anchor, freeze the downstream kernels and let the forecast be drawn from $\pi_\theta(\cdot | c)$ with c containing v ; then

$$\mathbb{E}[\nabla_\theta \log \pi_\theta(v | h) (W - \hat{V}(\hat{y}^{\text{tst}}))] = \mathbb{E}[\nabla_\theta \log \pi_\theta(v | h) W] - \nabla_\theta \mathbb{E}[\hat{V}(\hat{y}^{\text{tst}})],$$

the gradient on the right taken through $\pi_\theta(v | h)$ alone. The second term is a shaping term rather than a baseline. It favors reasoning that leads into the belief group of lower value, it vanishes when the two group values coincide, and its size is bounded by $|\hat{V}(\text{PASS}) - \hat{V}(\text{FAIL})|$ times the expected score norm. The scoring rule of Eq. 19 gives correct trusted PASS and FAIL forecasts with clean exits the same bonus, so the gap between the two group values is set by the batch rather than by the rule. The same reading applies to the decision-relative stream with the belief-decision values. Empirically, the trust rate rises during Stage II and the forecasts grow more accurate (§4.5).

C.3 Belief-augmented imitation

Stage I maximizes the likelihood of BeliefTraj under the step policy. Read as imitation, it controls the gap between the policy and the demonstration process.

Proposition 1 (Belief-augmented imitation bound). *Let P_E be the law of BeliefTraj trajectories over the corpus task distribution, and let π_θ act in the real environment on the same tasks. Write J for the task success probability, T for the step budget, and $H_E := \sum_{t=1}^T \mathbb{E}_{P_E}[H(P_E(\cdot | h_t))]$ for the conditional entropy of the demonstration process’s step messages $u_t = (v_t, \beta_t, a_t)$, reasoning included, $\varepsilon_{\text{SFT}}(\theta) := \mathcal{L}_{\text{ANTE-SFT}}(\theta) - H_E \geq 0$ for the excess Stage I loss, and $\Lambda := \sum_{t=1}^T \mathbb{E}_{P_E}[\text{TV}(P_E(o_{t+1} | h_t, u_t), P_{\text{env}}(o_{t+1} | h_t, a_t))]$ for the non-causal part of the demonstration process. Then*

$$J(P_E) - J(\pi_\theta) \leq \sqrt{T \varepsilon_{\text{SFT}}(\theta)/2} + \Lambda. \quad (2)$$

Proof. Since $R \in [0, 1]$ is a function of the trajectory, $J(P_E) - J(\pi_\theta) \leq \text{TV}(P_E, P_{\pi_\theta})$. Both laws share the task distribution and factorize over the $2T$ conditionals $u_t | h_t$ and $o_{t+1} | (h_t, u_t)$, with $P_E(\cdot | h_t)$ and $P_E(o_{t+1} | h_t, u_t)$ on one side and $\pi_\theta(\cdot | h_t)$ and $P_{\text{env}}(o_{t+1} | h_t, a_t)$ on the other, the environment depending on the step message only through a_t . Let $P^{(k)}$ be the law that takes the first k conditionals from P_E and the remaining ones from π_θ and the environment, so $P^{(2T)} = P_E$ and $P^{(0)} = P_{\pi_\theta}$. Consecutive hybrids differ in one conditional, share the distribution of its prefix under P_E , and share every later kernel; since adjoining a common continuation kernel while retaining the prefix preserves total variation, $\text{TV}(P^{(k)}, P^{(k-1)})$ equals the expected total variation of the k -th conditional under the P_E prefix. The triangle inequality over k gives

$$\text{TV}(P_E, P_{\pi_\theta}) \leq \sum_{t=1}^T \mathbb{E}_{P_E}[\text{TV}(P_E(\cdot | h_t), \pi_\theta(\cdot | h_t))] + \Lambda.$$

For the first sum, Pinsker’s inequality and Jensen’s inequality for the concave square root give $\mathbb{E}[\text{TV}] \leq \mathbb{E}[\sqrt{\text{KL}/2}] \leq \sqrt{\varepsilon_t/2}$ with $\varepsilon_t := \mathbb{E}_{P_E}[\text{KL}(P_E(\cdot | h_t) \| \pi_\theta(\cdot | h_t))] = \mathbb{E}_{P_E}[-\log \pi_\theta(u_t | h_t)] - \mathbb{E}_{P_E}[H(P_E(\cdot | h_t))]$, the step- t excess cross-entropy of the Stage I loss (§3.2), whose sum over t is $\varepsilon_{\text{SFT}}(\theta)$. Finally $\sum_t \sqrt{\varepsilon_t/2} \leq \sqrt{T \sum_t \varepsilon_t/2}$ by the Cauchy–Schwarz inequality. $\square$

The bound of Proposition 1 separates what Stage I optimizes from what it cannot touch. The first term is driven down by the likelihood objective, but only on the demonstration process’s own histories. The second term is a property of the corpus and of the environment. BeliefTraj writes the parsed future outcome back into β_t , so under P_E the next observation given (h_t, u_t) is supported only on observations consistent with that label. The environment law given (h_t, a_t) instead mixes over the posterior of the latent repository state. The two agree only when the history and the action already determine the outcome of the step, and Λ (Proposition 1) measures the information the hindsight labels carry beyond the history and the action; it is zero exactly when the two observation kernels agree P_E -almost surely at every step. For BeliefTraj, $J(P_E) = 1$ because the corpus contains solving trajectories only. The bound then reads as a statement about what can separate the fine-tuned policy from that ceiling: the corpus term, with the compounding term on top of it. This is the formal counterpart of §3.2, where Stage I leaves the prediction factor calibrated to corpus histories. Stage II, whose data are drawn causally from the environment at the policy’s own histories, addresses that gap.

C.4 Task reward under a world-model accuracy budget

The policy term of Eq. 7 is the clipped, token-level objective motivated by the standard step-level surrogate (Schulman et al., 2015, 2017), with the group baseline of §3.3 in place of a learned value function (Shao et al., 2024). Up to the distribution-shift term of the lemma, the surrogate lower-bounds the performance of the updated policy.

Lemma 1 (Step-level surrogate bound). *For step policies π and π' with $\pi'(\cdot | h) \ll \pi(\cdot | h)$ for every history h (as holds for full-support softmax policies), define the value functions of π recursively by $V^\pi(h_{T+1}) := R$, $Q^\pi(h_t, \beta, a) := \mathbb{E}_{o_{t+1} \sim P_{\text{env}}(\cdot | h_t, a)}[V^\pi(h_t, \beta, a, o_{t+1})]$ and $V^\pi(h_t) := \mathbb{E}_{(\beta, a) \sim \pi(\cdot | h_t)}[Q^\pi(h_t, \beta, a)]$, so that Q^π is*

defined for every emission, and let $A^\pi := Q^\pi - V^\pi \in [-1, 1]$. Define the surrogate

$$\begin{aligned} L_\pi(\pi') &:= J(\pi) + \sum_{t=1}^T \mathbb{E}_{h_t \sim P_\pi} \mathbb{E}_{(\beta, a) \sim \pi'(\cdot | h_t)} [A^\pi(h_t, \beta, a)] \\ &= J(\pi) + \sum_{t=1}^T \mathbb{E}_{h_t \sim P_\pi} \mathbb{E}_{(\beta, a) \sim \pi(\cdot | h_t)} \left[\frac{\pi'(\beta, a | h_t)}{\pi(\beta, a | h_t)} A^\pi(h_t, \beta, a) \right], \end{aligned} \quad (12)$$

the second form being the joint-emission importance-sampling representation that underlies the token-level clipped estimator in the policy term of Eq. 7. With $\Delta_{\text{TV}} := \sup_h \text{TV}(\pi'(\cdot | h), \pi(\cdot | h))$ and $\Delta_{\text{KL}} := \sup_h \text{KL}(\pi(\cdot | h) \parallel \pi'(\cdot | h))$,

$$J(\pi') \geq L_\pi(\pi') - T(T-1) \Delta_{\text{TV}} \geq L_\pi(\pi') - T(T-1) \sqrt{\Delta_{\text{KL}}/2}. \quad (13)$$

Proof. By the recursive definitions, $\mathbb{E}[V^\pi(h_{t+1}) - Q^\pi(h_t, \beta_t, a_t) | h_t, \beta_t, a_t] = 0$ under either policy, because the observation is drawn from the shared kernel P_{env} , and $\mathbb{E}_{P_{\pi'}}[V^\pi(h_1)] = J(\pi)$ because h_1 is the task alone. Telescoping $R - V^\pi(h_1) = \sum_{t=1}^T (V^\pi(h_{t+1}) - V^\pi(h_t))$ along a trajectory of π' and taking expectations gives the performance difference identity $J(\pi') - J(\pi) = \sum_{t=1}^T \mathbb{E}_{h_t \sim P_{\pi'}} \mathbb{E}_{(\beta, a) \sim \pi'(\cdot | h_t)} [A^\pi(h_t, \beta, a)]$ (Kakade and Langford, 2002). Writing $g_t(h) := \mathbb{E}_{(\beta, a) \sim \pi'(\cdot | h)} [A^\pi(h, \beta, a)] \in [-1, 1]$, the surrogate differs from the identity only in the law of h_t , so $|J(\pi') - L_\pi(\pi')| \leq \sum_{t=1}^T 2 \text{TV}(P_{\pi'}(h_t), P_\pi(h_t))$. The two laws of h_t share the task distribution and all environment kernels and differ in the $t-1$ policy conditionals before step t ; the hybrid argument of Proposition 1 bounds their total variation by $\sum_{s < t} \mathbb{E}_{P_\pi} [\text{TV}(\pi'(\cdot | h_s), \pi(\cdot | h_s))] \leq (t-1) \Delta_{\text{TV}}$. Summing, $|J(\pi') - L_\pi(\pi')| \leq 2 \Delta_{\text{TV}} \sum_{t=1}^T (t-1) = T(T-1) \Delta_{\text{TV}}$, and Pinsker's inequality gives the second form. $\square$

The policy term differs from the surrogate of Eq. 12 in three ways. The ratio is factorized over tokens rather than taken per emission. The advantage is a group-baseline estimate from the task's own rollouts rather than A^π . The return is the shaped one of the repository instantiation, combining task outcome, bookkeeping term, and decision rewards; its width is at most $1.4 + T(\alpha + \max(\gamma, \kappa))$ in place of the unit range in the constant of Eq. 13, and the decision return is what BAPO measures against belief-conditional baselines (Proposition 3). The clip range $(\epsilon_{\text{lo}}, \epsilon_{\text{hi}})$ of Eq. 7 plays the role of the trust region, as in PPO (Schulman et al., 2017). The remaining question is what the prediction term does to the optimum.

Proposition 4 (Multiplier as accuracy budget). *Fix $\lambda \geq 0$, let $\bar{\mathcal{L}}$ be the expected prediction loss of Eq. 8 averaged over supervised steps, and suppose that for every multiplier considered the scalarized objective $J(\theta) - \lambda \bar{\mathcal{L}}(\theta)$ attains a global maximum θ_λ with finite expected prediction loss. Then θ_λ is optimal for the constrained program*

$$\begin{aligned} \max_{\theta} \quad & J(\theta) \\ \text{s.t.} \quad & \bar{\mathcal{L}}(\theta) \leq B(\lambda), \end{aligned}$$

where $B(\lambda) := \bar{\mathcal{L}}(\theta_\lambda)$, and the budget is monotone: for $\lambda' > \lambda$, $B(\lambda') \leq B(\lambda)$ and $J(\theta_{\lambda'}) \leq J(\theta_\lambda)$.

Proof. If some θ had $\bar{\mathcal{L}}(\theta) \leq \bar{\mathcal{L}}(\theta_\lambda)$ and $J(\theta) > J(\theta_\lambda)$, then $J(\theta) - \lambda \bar{\mathcal{L}}(\theta) > J(\theta_\lambda) - \lambda \bar{\mathcal{L}}(\theta_\lambda)$, contradicting the optimality of θ_λ . For monotonicity, optimality of θ_λ and of $\theta_{\lambda'}$ for their own multipliers gives $J(\theta_\lambda) - \lambda \bar{\mathcal{L}}(\theta_\lambda) \geq J(\theta_{\lambda'}) - \lambda \bar{\mathcal{L}}(\theta_{\lambda'})$ and $J(\theta_{\lambda'}) - \lambda' \bar{\mathcal{L}}(\theta_{\lambda'}) \geq J(\theta_\lambda) - \lambda' \bar{\mathcal{L}}(\theta_\lambda)$; adding them yields $(\lambda' - \lambda)(\bar{\mathcal{L}}(\theta_\lambda) - \bar{\mathcal{L}}(\theta_{\lambda'})) \geq 0$, hence $\bar{\mathcal{L}}(\theta_{\lambda'}) \leq \bar{\mathcal{L}}(\theta_\lambda)$, and substituting into the first inequality gives $J(\theta_\lambda) - J(\theta_{\lambda'}) \geq \lambda(\bar{\mathcal{L}}(\theta_\lambda) - \bar{\mathcal{L}}(\theta_{\lambda'})) \geq 0$. $\square$

Training with a frozen multiplier is the special case of reward-constrained policy optimization (Altman, 1999; Tessler et al., 2019): the multiplier indexes a monotone family of accuracy budgets.

C.5 Information-theoretic guarantees of the prediction loss

Proposition 5 relates the expected prediction loss of Eq. 8 to what the belief knows. Corollaries 1 and 2 then show that the forecast divergence bounds the error of a decision reward evaluated under the belief. Proposition 2, proved here, and Corollary 3 bound the regret of choosing among candidates by their forecasts.

Proposition 5 (Prediction-loss information bounds). *For every supervised step,*

$$H(y_t | c_t) \leq \bar{\mathcal{L}}_t(\theta), \quad I(c_t; y_t | a_t) \geq H(y_t | a_t) - \bar{\mathcal{L}}_t(\theta), \quad (14)$$

and

$$I(c_t; y_t | a_t) \leq I(c_t; o_{t+1} | a_t) \leq I(c_t; s_{t+1} | a_t), \quad (15)$$

and, for the written part of the context, the reasoning and the summary together,

$$I((v_t, z_t); y_t | h_t) \geq H(y_t | h_t) - \bar{\mathcal{L}}_t(\theta). \quad (16)$$

Proof. For any conditional distribution $q(\cdot | c)$, $\mathbb{E}[-\log q(y_t | c_t)] = H(y_t | c_t) + \mathbb{E}[\text{KL}(p(\cdot | c_t) \| q(\cdot | c_t))] \geq H(y_t | c_t)$, the variational bound of [Barber and Agakov \(2003\)](#); with $q = q_\theta$ this is Eq. 8. Conditioning reduces entropy, so $H(y_t | c_t, a_t) \leq H(y_t | c_t) \leq \bar{\mathcal{L}}_t(\theta)$, and $I(c_t; y_t | a_t) = H(y_t | a_t) - H(y_t | c_t, a_t)$ gives the second inequality of Eq. 14. For Eq. 15: the context c_t is formed before o_{t+1} exists; given (s_{t+1}, a_t) the observation is drawn from $\Omega(\cdot | s_{t+1}, a_t)$ independently of c_t (under supervised-step conditioning the kernel is restricted to verifier-valid observations, and since parsing depends on the emitted message and validity on (o_{t+1}, a_t) , the independence is preserved), and given (o_{t+1}, a_t) the outcome $y_t = \psi(o_{t+1}, \chi(a_t))$ is deterministic, so conditionally on a_t the chain $c_t \rightarrow s_{t+1} \rightarrow o_{t+1} \rightarrow y_t$ is Markov and both inequalities are the conditional data processing inequality. For Eq. 16, $H(y_t | h_t, v_t, z_t) = H(y_t | c_t) \leq \bar{\mathcal{L}}_t(\theta)$ and $I((v_t, z_t); y_t | h_t) = H(y_t | h_t) - H(y_t | h_t, v_t, z_t)$. $\square$

The expected prediction loss is an upper bound on $H(y_t | c_t)$, the uncertainty about the checkable outcome that remains at the moment the block is written. Minimizing it therefore lowers the forecast divergence at a fixed outcome law and, through the policy’s own choices of context and action, bounds the uncertainty left at writing time. In turn, $H(y_t | a_t) - \bar{\mathcal{L}}_t$ is a lower bound on the information the writing context carries about the outcome of the committed action, hence, by Eq. 15, about the latent state that action leads to. The conditioning on a_t is what makes the statement about the state rather than about the action: unconditionally, the context is trivially informative about the action the policy is about to take. Eqs. 14 and 15 are about the context c_t in which the value tokens are generated; the predictor reads the whole history, and the belief is the policy’s own summary of that history rather than a separate module. Eq. 16 isolates the written part of the context. Whatever the expected prediction loss leaves of $H(y_t | h_t)$, the uncertainty that the raw history alone leaves about the checkable outcome, is information carried by the reasoning and the written summary, including what they induce through the chosen action. This is the objective used to train belief modules for imitation learning in POMDPs ([Gangwani et al., 2020](#)), with two changes: the belief lives in the policy’s emitted tokens, and the target is a parsed, verifiable outcome rather than the raw observation, which removes the need for a learned decoder. The classical reason for wanting the context to be informative about the latent state is that optimal POMDP policies act on beliefs, sufficient statistics of the history ([Åström, 1965](#); [Kaelbling et al., 1998](#)) (Figure 7). The forecast divergence $\mathbb{E}[\delta(c_t)] = \bar{\mathcal{L}}_t - H(y_t | c_t)$ is the part of the expected prediction loss the objective can remove; the entropy term is set by the environment and by the policy’s own action distribution.

Corollary 1 (Decision-reward error bound). *Fix a supervised step with writing context c and predictive $q = q_\theta(\cdot | c)$. Let $p = p(\cdot | c)$ be the outcome law at the writing context, before the belief is committed, and $p_{\hat{y}, d} = p(\cdot | c, \hat{y}, d)$ the law under which the decision reward is realized once the belief $(\hat{y}, d)$, drawn from the policy’s law at c under $\mathcal{M}$, has been written and acted on, all laws taken under the same conditioning as δ and I . Write $\delta = \text{KL}(p \| q)$ for the forecast divergence and $I := I((\hat{y}, d); y | c)$ for the information the committed belief carries about the outcome beyond the context. For any payoff $u : \mathcal{Y} \rightarrow \mathbb{R}$ with $\|u\|_\infty \leq U$,*

$$|\mathbb{E}_p[u] - \mathbb{E}_q[u]| \leq U\sqrt{2\delta}, \quad \mathbb{E}_{(\hat{y}, d)} |\mathbb{E}_{p_{\hat{y}, d}}[u] - \mathbb{E}_q[u]| \leq U\sqrt{2(\delta + I)}.$$

In particular, averaged over the committed beliefs, the expected decision reward (Eq. 19, payoffs bounded by $\alpha \vee \gamma \vee \kappa$ and zero on inconclusive outcomes) evaluated under the belief differs from its expectation under the realized law by at most $(\alpha \vee \gamma \vee \kappa)\sqrt{2(\delta + I)}$.

The prediction loss controls δ , while I is zero when the belief is a deterministic function of the context and positive whenever the sampled belief steers the action toward different outcomes; it is at most the action-information term of Eq. 18, with equality when the action adds no information about the outcome beyond the belief.

Proof. For the marginal law, $|\mathbb{E}_p[u] - \mathbb{E}_q[u]| \leq \|u\|_\infty \sum_y |p(y) - q(y)| \leq 2U \text{TV}(p, q) \leq U \sqrt{2 \text{KL}(p \| q)}$ by Pinsker's inequality. For the realized law, the chain rule of relative entropy gives $\mathbb{E}_{(\hat{y}, d)} [\text{KL}(p_{\hat{y}, d} \| q)] = \mathbb{E}_{(\hat{y}, d)} [\text{KL}(p_{\hat{y}, d} \| p)] + \text{KL}(p \| q) = I + \delta$, since q does not depend on $(\hat{y}, d)$ and $\mathbb{E}_{(\hat{y}, d)} [p_{\hat{y}, d}] = p$; Pinsker applied to each $p_{\hat{y}, d}$ and Jensen's inequality for the concave square root give the second bound. $\square$

Corollary 2 (Trajectory-level cost bound). *Fix a rollout i and a nonempty set S of supervised steps at which predictions are consumed, with forecast divergences δ_t for $t \in S$ and average $\bar{\delta} := |S|^{-1} \sum_{t \in S} \delta_t$. For per-step payoffs with $\|u_t\|_\infty \leq U$,*

$$\sum_{t \in S} |\mathbb{E}_{p_t}[u_t] - \mathbb{E}_{q_t}[u_t]| \leq U \sum_{t \in S} \sqrt{2\delta_t} \leq |S| U \sqrt{2\bar{\delta}}.$$

The cost of consuming predictions along a rollout is therefore controlled by the average forecast divergence over the steps at which predictions are consumed, rather than by its worst case. Averaged over the commitments at each step, the same bound holds under the committed-belief laws with $\delta_t + I_t$ in place of δ_t , where I_t is the information term of Corollary 1 at step t .

Proof. Apply Corollary 1 at each step of S and sum; the second inequality is Jensen's inequality for the concave square root. $\square$

Proposition 2 (Prediction-to-decision regret bound). *Let candidates $k = 0, \dots, K$ at one step have outcome laws p_k under execution and belief predictives q_k , with $\delta_k := \text{KL}(p_k \| q_k)$ the divergence of each forecast from its executed outcome law. For a score $u : \mathcal{Y} \rightarrow [0, U]$, let k^* maximize the executed value $v_k := \mathbb{E}_{p_k}[u]$ and $\hat{k}$ the believed value $\hat{v}_k := \mathbb{E}_{q_k}[u]$. Then*

$$v_{k^*} - v_{\hat{k}} \leq U(\sqrt{\delta_{k^*}/2} + \sqrt{\delta_{\hat{k}}/2}) \leq U \sqrt{2 \max_k \delta_k}. \quad (3)$$

Proof of Proposition 2. Write $c^{(k)}$ for the writing context of candidate k , so that $q_k = q_\theta(\cdot | c^{(k)})$ and $p_k = p(\cdot | h, a^{(k)})$. Since $\hat{v}_{\hat{k}} \geq \hat{v}_{k^*}$, $v_{k^*} - v_{\hat{k}} \leq (v_{k^*} - \hat{v}_{k^*}) + (\hat{v}_{\hat{k}} - v_{\hat{k}})$, and since u takes values in $[0, U]$ each bracket is at most $U \text{TV}(p_k, q_k) \leq U \sqrt{\delta_k/2}$ by Pinsker's inequality, applied to candidate k^* and to candidate $\hat{k}$. $\square$

Corollary 3 (Point-forecast regret bound). *In the setting of Proposition 2, let each candidate be scored by a single point forecast, $\tilde{v}_k := u(\hat{y}^{(k)})$ with $\hat{y}^{(k)}$ the forecast the procedure emits for candidate k , and let $\bar{k} \in \arg \max_k \tilde{v}_k$. With $\zeta := \mathbb{E}[\max_k |u(\hat{y}^{(k)}) - \hat{v}_k|]$ the expected largest deviation of a drawn score from the believed value,*

$$\mathbb{E}[v_{k^*} - v_{\bar{k}}] \leq U \mathbb{E}[\sqrt{2 \max_k \delta_k}] + 2\zeta, \quad (17)$$

the expectation taken over the generation of the candidates, under which the δ_k are random.

The second term is the outcome spread at uncertain steps: it is zero when every q_k puts all its mass on one score level, and it need not vanish as the forecast divergences δ_k vanish, since a candidate whose outcome is genuinely uncertain can be outscored by a single draw even under a perfect predictive.

Proof. Write $v_{k^*} - v_{\bar{k}} = (v_{k^*} - \hat{v}_{k^*}) + (\hat{v}_{k^*} - \tilde{v}_{k^*}) + (\tilde{v}_{k^*} - \tilde{v}_{\bar{k}}) + (\tilde{v}_{\bar{k}} - \hat{v}_{\bar{k}}) + (\hat{v}_{\bar{k}} - v_{\bar{k}})$. The third bracket is at most zero by the choice of $\bar{k}$; the first and last are bounded as in Proposition 2; the second and fourth are each at most $\max_k |u(\hat{y}^{(k)}) - \hat{v}_k|$ pointwise, and taking expectations gives 2ζ . $\square$

Two points connect Proposition 2 to the training procedure. First, the forecast divergence of a candidate is measured against the outcome of its own action, whereas the predictive was written before that action. The two are tied by an identity that holds because the outcome is independent of the written reasoning and summary given the history and the action,

$$\mathbb{E}_{a \sim P(\cdot | c, \mathcal{M})} [\text{KL}(p(\cdot | h, a) \| q_\theta(\cdot | c))] = \delta(c) + I(a_t; y_t | c_t = c), \quad (18)$$

so, in expectation over the action distribution, the candidates’ forecast divergences are small when the trained forecast divergence is small and the writing context already determines what the committed action will do; the second term is the cost of predicting before acting. Second, ANTE-TTO (§3.4, Algorithm 3, $U = 4$ for the score of Eq. 23) scores each candidate by the plug-in $\hat{v}_k$ of Corollary 3, with the decoder greedy for the original continuation and sampling at temperature one for the alternatives, and with ζ taken under that decoder, so its regret obeys Eq. 17. The bound says that the part of the regret attributable to the world model is controlled by the forecast divergence. A smaller candidate divergence tightens the bound.

Instantiation across harnesses. Every statement above depends on the environment through the action-type map χ , the verifier ψ of Appendix A, and the reward the harness assigns. On repository issues ψ reads the check field and the exit field, and no constant in the propositions references the harness otherwise. The terminal instantiation (Appendix I) keeps the exit field as the per-step coordinate that ψ reads from the next observation and anchors the belief groups on its forecast, so the per-step statements apply there to that coordinate and to rewards that depend on it, with the terminal verifier and scoring rule in place.

D Corpus curation

BeliefTraj is a corpus of solving trajectories collected by a capable agent in the execution environment, with its forecast labels read from execution. The paired corpora for Reactive and Retrospective use the same trajectories, without the belief block or with only its summary. This section covers task screening, label derivation and its checks, the paired construction with its protocol-leakage filter, and the corpus statistics behind Table 1. Figure 8 maps the pipeline, and Table 5 provides the stage counts.

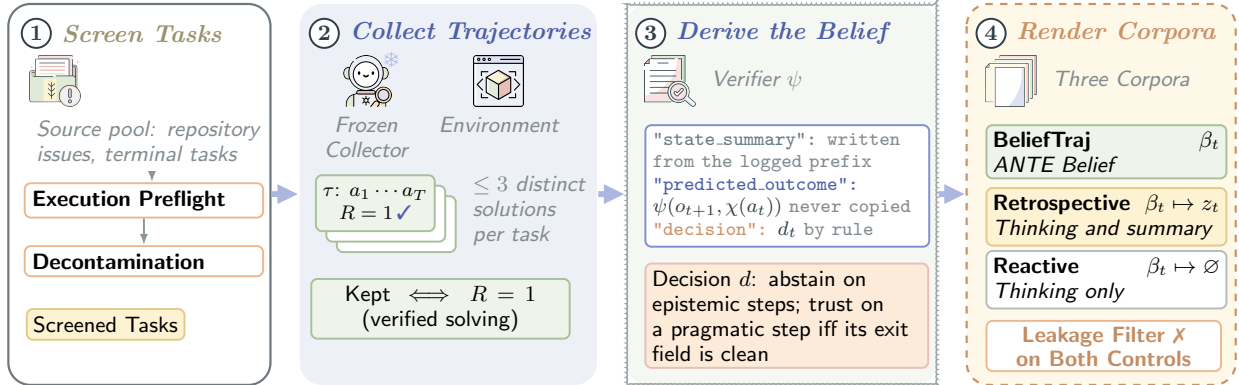


Figure 8 BeliefTraj curation. Screening, collection in containers, and label derivation; counts in Table 5.

Table 5 Task screening counts.

Stage	Count
Source pool of repository-issue tasks	13,558
Reproduce in their containers (execution preflight)	10,662
Solved by the trajectory collector	6,746
Verified solving trajectories (quality gates)	12,500 over 6,729 tasks
Training corpora (ANTE, Retrospective, Reactive)	$3 \times 12,500$ trajectories

D.1 Task screening and trajectory collection

Before any model call, the source pool (Table 5) passes an execution preflight, in which every task must reproduce in its own container, and decontamination against the evaluation benchmarks. A strong collector (an LLM such as Qwen3-Coder-480B-A35B (Qwen Team, 2025) or DeepSeek-V3.2 (DeepSeek-AI, 2025)) then attempts the tasks in the containers, and only verified solving trajectories survive: BeliefTraj keeps 12,500 of them over 6,729 repository tasks and 30,161 over 10,823 terminal tasks. Table 6 gives the composition of the three source pools, R2E-Gym (Jain et al., 2025), SWE-rebench (Badertdinov et al., 2025), and SWE-Gym (Pan et al., 2025), including the RL pool of 6,566 unique instances.

Table 6 RL pool composition. Per-pool composition of the repository-issue tasks after the execution preflight.

Pool	Executable tasks	Solved in collection	Solve rate	In RL pool
R2E-Gym	4,555	3,919	86.0%	3,880
SWE-rebench	4,456	1,829	41.0%	1,767
SWE-Gym	1,651	998	60.4%	919
Total	10,662	6,746	63.3%	6,566

Step 1. Thinking: “Let me explore the repo structure to find the URL dispatcher code.”

Reactive

—

Retrospective

```
<memory_state>{"state_summary": "Starting to investigate the aiohttp URL dispatcher issue where routes with variables after a fixed path segment fail to match."}</memory_state>
```

ANTE (BeliefTraj)

```
<belief_state>{"state_summary": "Starting to investigate the aiohttp URL dispatcher issue where routes with variables after a fixed path segment fail to match.", "predicted_outcome": {"target_test": "NA", "exit_ok": true}, "decision": "trust"}</belief_state>
```

Tool call: <function=search> def index_key · /testbed/aiohttp/web_urldispatcher.py
... Steps 2–4 ...

Step 5. Thinking: “Let me run this to confirm the failure.”

Reactive

—

Retrospective

```
<memory_state>{"state_summary": "Created reproduce_issue.py to confirm the 404 for /core/locations/tail/here."}</memory_state>
```

ANTE (BeliefTraj)

```
<belief_state>{"state_summary": "Created reproduce_issue.py to confirm the 404 for /core/locations/tail/here.", "predicted_outcome": {"target_test": "FAIL", "exit_ok": true}, "decision": "abstain"}</belief_state>
```

Tool call: <function=execute_bash> cd /testbed && python reproduce_issue.py
... Steps 6–14 ...

Step 15 (Submission). Thinking: “The task is complete. The fix resolves the issue and is verified. ...”

Reactive

—

Retrospective

```
<memory_state>{"state_summary": "Confirmed the 3 static-permission test failures are pre-existing (fail on unmodified code too, since tests run as root); my index-key fix resolves the issue with all matching tests passing and the reproduce script printing VERDICT: PASS."}</memory_state>
```

ANTE (BeliefTraj)

```
<belief_state>{"state_summary": "Confirmed the 3 static-permission test failures are pre-existing (fail on unmodified code too, since tests run as root); my index-key fix resolves the issue with all matching tests passing and the reproduce script printing VERDICT: PASS.", "predicted_outcome": {"target_test": "PASS", "exit_ok": true}, "decision": "trust"}</belief_state>
```

Tool call: <function=finish> submit · "Fixed the URL dispatcher's route indexing in /testbed/aiohttp/web_urldispatcher.py."
..."

Figure 9 Trajectory samples from the SFT corpora. Steps 1, 5, and 15 (the submission) of one recorded run, with long passages elided.

D.2 Execution-derived labels and checks

Label derivation. The verifier ψ re-derives the prediction labels from the logged observation, overriding the collector’s own claim on 59,212 check-field and 17,407 exit-field labels.

Decision labels. The decision labels follow a fixed rule: $d_t = \text{ABSTAIN}$ on every epistemic step, whose check is still open, and TRUST on pragmatic steps exactly when the re-derived exit field is clean. A demonstration carries no counterfactual for the untaken decision, so the label fixes the format and leaves calibration to Stage II (§4.5).

Reward monitoring: test-tampering checks. We monitor test-file activity during training with an over-inclusive screen that counts every command touching a test file or a test entry point. The screen is observational and never enters the reward. None of the 2,442 commands it catches is reward hacking, and the task reward comes from hidden tests in a fresh container the agent cannot reach.

D.3 Paired corpus construction and protocol-leakage filtering

Corpus construction. Each corpus trajectory is written three times from one recorded run, sharing the trajectory’s tool calls and, apart from the cleaning below, its reasoning (Figure 9): Reactive carries no block, Retrospective the summary alone in a `<memory_state>` block, and ANTE (BeliefTraj) the summary, the

predicted outcome, and the decision. The two control corpora then pass a protocol-leakage filter: reasoning that references the belief protocol, its prediction fields, or the decision vocabulary is removed or rewritten, so that neither control variant sees the protocol it is meant to ablate. The block is inserted at the same position in every assistant step, immediately before the tool call, so the variants differ by the protocol block and by this cleaning alone.

Decontamination. Decontamination finds no overlap with SWE-bench Verified at the task and repository levels, and, for the RL pool, none with SWE-bench Lite at the instance, owner/repository, and R2E-Gym-repository levels.

D.4 Corpus statistics and evaluation task lists

Per-task solution counts. Up to three distinct verified solutions are kept per task. On the repository side the split is 1,003 tasks with one solution, 5,681 with two, and 45 with three; on the terminal side 785, 738, and 9,300 respectively.

Step statistics. Figure 10 profiles the repository training corpus (12,500 trajectories). Trajectories average 11,962 tokens under the training tokenizer (median 10,073, p95 26,023, max 32,666, inside the 32k window). Every assistant step carries exactly one parsed belief state. Steps that predict a definite check outcome make up 18.5% of assistant steps (42,676: 26,751 PASS and 15,925 FAIL). Of these, the 30,178 test steps whose observation the verifier evaluated (13.1%) are the conclusive steps, where the decision is scored; the rest are submission steps, whose check field keeps the collector’s verdict. The action-type map χ marks 47,997 steps epistemic (20.8%).

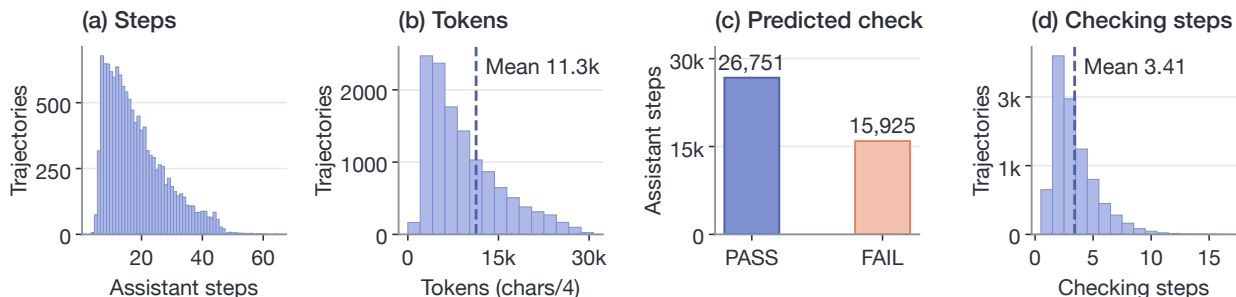


Figure 10 Training corpus distributions. (a) Assistant steps per trajectory. (b) Tokens per trajectory (characters/4 estimate). (c) Predicted PASS and FAIL counts. (d) Steps with a definite predicted check outcome, per trajectory.

Terminal evaluation sets. We build TB-Dev following the Internal-dev recipe of ECHO (Shrivastava et al., 2026). TB-Dev collects the 73 data-processing, systems-operations, and development-tooling tasks from Terminal-Bench 1.0 (The Terminal-Bench Team, 2025) and OpenThoughts-TB-dev (OpenThoughts-Agent team, 2025) that share neither a name nor a near-duplicate instruction with Terminal-Bench 2.0 or TB-Lite.

E Belief state specification and outcome evaluation

This section introduces the details of the belief block’s schema, the map from a response to an outcome, and the decision reward for each harness.

E.1 Fields of the belief state

The belief is emitted as a tagged JSON block immediately before the tool call, in the same assistant message (Figure 11). `state_summary` carries the summary z_t ; `predicted_outcome.target_test` is $\hat{y}_t^{\text{tst}}$; `predicted_outcome.exit_ok` is $\hat{y}_t^{\text{exe}}$; `decision` is d_t . Figure 9 shows a pragmatic step, $\hat{y} = (\text{NA}, 1)$, $d = \text{TRUST}$, and an epistemic step, $\hat{y} = (\text{FAIL}, 1)$, $d = \text{ABSTAIN}$.

The epistemic step predicts FAIL for a reproduction script, since the bug is expected to be present, and chooses to abstain. Within the block, the two predicted-value tokens receive $\mathcal{L}_{\text{pred}}$ and the decision token the policy gradient.

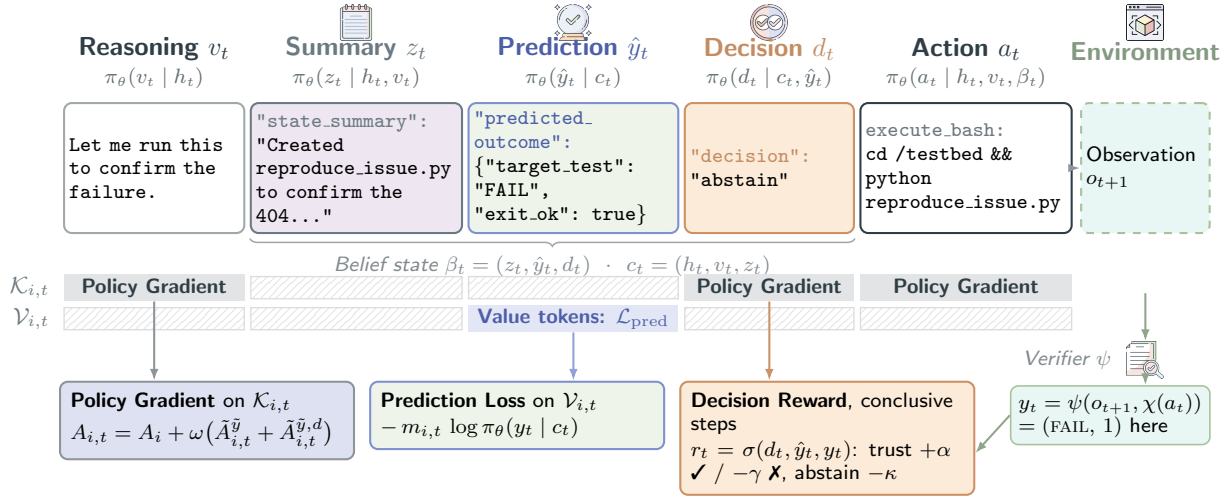


Figure 11 Anatomy of a step message. The belief state $\beta_t = (z_t, \hat{y}_t, d_t)$ is between the reasoning and the tool call in one pass. The prediction loss reaches the forecast’s value tokens, and the policy gradient reaches the reasoning, decision, and action tokens. The verifier ψ maps o_{t+1} to the realized outcome y_t , which checks the forecast and settles the decision.

E.2 Outcome channels, parsing rules, and decision reward per harness

Action typing and the verifier. The action-type map χ of §3.1 marks as epistemic the steps that run a check the verifier can evaluate, and as pragmatic all remaining steps. Its syntactic form is the EPISTEMIC/PRAGMATIC rule of the belief protocol in Appendix O. The check field is instantiated per scaffold (Table 7). On the repository scaffold it is the outcome of the check the step runs; on pragmatic steps it is NA by definition, which leaves the collector’s forecast unsupervised there. On the terminal scaffold it is the outcome of the hidden task verifier (the `expect_success` target), which every step carries. On the terminal corpus the predicted-outcome key set is $\{\text{exit_ok}, \text{expect_success}\}$; steps carrying a predicted `target_test`, for which no grading source exists, are dropped at corpus construction. The block schema is identical at collection, fine-tuning, reinforcement learning, and evaluation.

Table 7 Parser rules. The check field and the exit field per scaffold.

	Repository issues (R2E-Gym)	Terminal tasks
Check field predicts	Outcome of the check the step runs	Outcome of the hidden task verifier
Values	PASS / FAIL / NA	PASS / FAIL
NA is definitional on	Pragmatic steps	–
Infeasible pair	(PASS, exit field 0): a passing test implies a clean run	–
Exit field predicts	Command exits with code zero	Same
Conclusive iff	Check field in {PASS, FAIL}	After the last step
Anchor key	Predicted check field	Predicted exit field

Scoring and shaping details. On conclusive steps, where $y_{i,t}^{\text{tst}} \in \{\text{PASS}, \text{FAIL}\}$, the scoring rule σ of §3.3 is

$$r_{i,t} = \sigma(d_{i,t}, \hat{y}_{i,t}, y_{i,t}) = \begin{cases} +\alpha & d_{i,t} = \text{TRUST}, \hat{y}_{i,t} = y_{i,t}, y_{i,t}^{\text{exe}} = 1, \\ 0 & d_{i,t} = \text{TRUST}, \hat{y}_{i,t} = y_{i,t}, y_{i,t}^{\text{exe}} = 0, \\ -\gamma & d_{i,t} = \text{TRUST}, \hat{y}_{i,t} \neq y_{i,t}, \\ -\kappa & d_{i,t} = \text{ABSTAIN}, \end{cases} \quad (19)$$

where $\alpha, \gamma, \kappa > 0$ and equality is vector equality over both fields. Every inconclusive epistemic step scores zero, and so does a fully correct trusted crash forecast, which is also excluded from the anchor groups. Two asymmetries hold on the repository scaffold. A trusted NA on a step the verifier decides counts as a miss, so a predicted NA never scores $+\alpha$. Agreement between a predicted and a realized NA is rewarded only through

Algorithm 2 ANTE pipeline, expanded from Algorithm 1

Require: pool $\mathcal{D}$; N ; π_{ref} ; ψ , χ ; (α, γ, κ) ; ω , λ , η , η_{lr}
// BeliefTraj curation
 1: collect verified solving trajectories in the containers
 2: $y_{i,t} \leftarrow \psi(o_{i,t+1}, \chi(a_{i,t}))$, written back as $\hat{y}_{i,t}$; label $d_{i,t}$ by rule
// Stage I: supervised fine-tuning on BeliefTraj
 3: $\theta \leftarrow \arg \min_{\theta}$ next-token CE on BeliefTraj
// Stage II: reinforcement learning with BAPO
 4: **for** each iteration **do**
 5: sample $x \sim \mathcal{D}$; roll out $\tau_1, \dots, \tau_N \sim \pi_{\theta}(\cdot | x)$: each step emits $\beta_{i,t}$, then $a_{i,t}$, executed
 6: $y_{i,t} \leftarrow \psi(o_{i,t+1}, \chi(a_{i,t}))$
 7: $r_{i,t} \leftarrow \sigma(d_{i,t}, \hat{y}_{i,t}, y_{i,t})$ on $\mathcal{C}_i$, else 0 ▷ Eq. 19
 8: $A_i \leftarrow R_i - \frac{1}{N} \sum_j R_j$; $W_{i,t} \leftarrow \sum_{s \geq t} r_{i,s}$
 9: $\hat{V}(\hat{y}) \leftarrow \text{mean}_{G(\hat{y})} W$; $\hat{Q}(\hat{y}, d) \leftarrow \text{mean}_{G(\hat{y}, d)} W$
 10: $\Delta_{i,t} \leftarrow \omega(\hat{A}_{i,t}^{\hat{y}} + \hat{A}_{i,t}^{\hat{y}, d})$ on anchored steps, else 0; $\Delta \equiv 0$ on a task with no trusted conclusive step
 11: $A_{i,t} \leftarrow A_i + \Delta_{i,t}$ ▷ Eq. 6
 12: $\theta \leftarrow \theta - \eta_{\text{lr}} \nabla_{\theta} \mathcal{L}(\theta)$ ▷ Eq. 7
 13: **end for**

the prediction loss of Eq. 4, whose value tokens receive no policy gradient, because NA is the one outcome the policy can produce at will (an obfuscated test command forces it).

F Training setup and infrastructure

F.1 Supervised fine-tuning records

Figure 12 shows the supervised fine-tuning loss of the four ANTE runs: 7B, 14B, and 32B on the repository scaffold and 9B on the terminal scaffold.

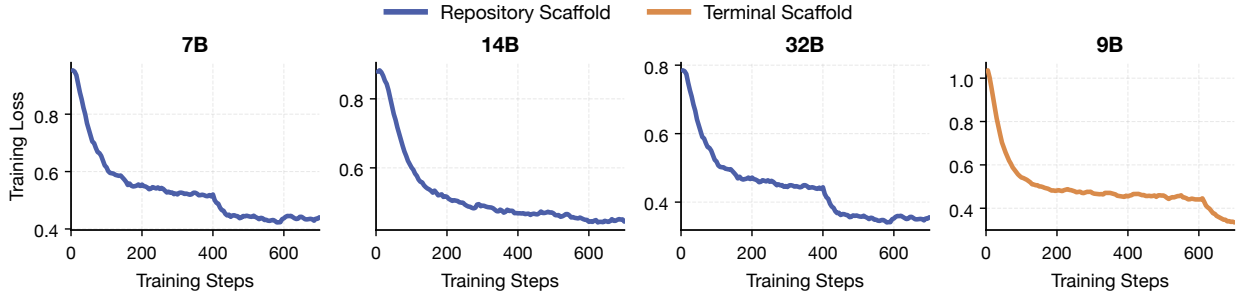


Figure 12 Supervised fine-tuning loss of ANTE.

F.2 The optimization objective

Group-relative objective. For a task x , the current policy samples N rollouts $\tau_1, \dots, \tau_N$, each receives its task reward R_i , and GRPO (Shao et al., 2024) maximizes the clipped surrogate

$$\mathcal{J}(\theta) = \mathbb{E} \left[\frac{1}{N} \sum_{i=1}^N \frac{1}{|\tau_i|} \sum_{k=1}^{|\tau_i|} \min(\varrho_{i,k} \hat{A}_i, \text{clip}(\varrho_{i,k}, 1 - \epsilon, 1 + \epsilon) \hat{A}_i) \right] - \eta D_{\text{KL}}(\pi_{\theta} \parallel \pi_{\text{ref}}), \quad (20)$$

where $\hat{A}_i = (R_i - \text{mean}_j R_j) / \text{std}_j R_j$ is the group-normalized advantage and $\varrho_{i,k} = \pi_{\theta} / \pi_{\text{old}}$ the importance ratio of the k -th generated token of τ_i . Dr.GRPO (Liu et al., 2025) drops the per-sequence length normalization and the standard-deviation scaling, and DAPO (Yu et al., 2025a) widens the upper clip to $\epsilon_{\text{hi}} > \epsilon_{\text{lo}}$; ANTE adopts both, keeps the token-mean aggregation, and adds the step-level signals of §3.3: the prediction loss, the decision reward, and the anchored advantages. Every member of this family assigns one advantage per rollout, broadcast to every token of every step; that is the property BAPO changes.

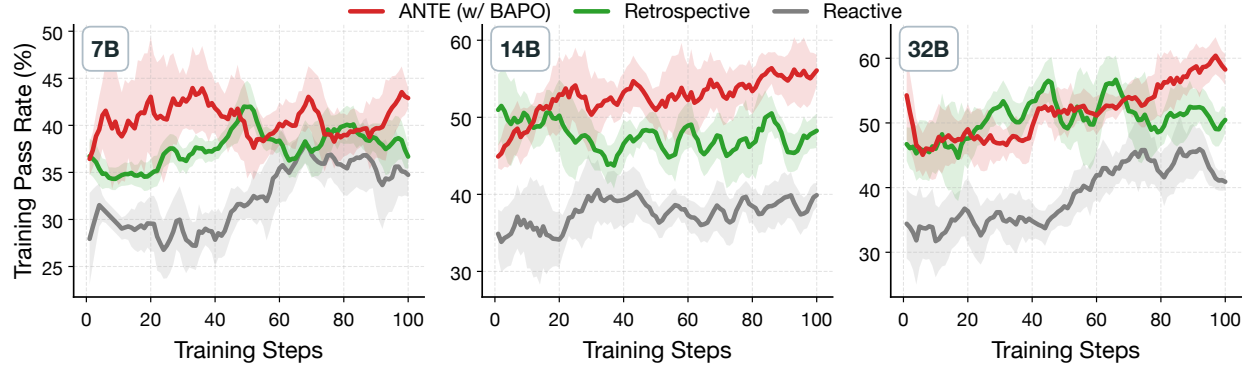


Figure 13 Training pass rate under RL. Training pass rate (%) for Reactive, Retrospective, and ANTE at 7B, 14B, and 32B on the RL task pool.

Table 8 lists the repository-task configuration of both training stages; the remaining terminal-task values that differ from it (rollouts per task, step limit, response length, and batch geometry) are listed in Appendix I.

Table 8 Training configuration. Settings are shared across model scales except where noted.

	Parameter	Value
SFT	Objective	Next-token cross-entropy
	Corpus	One variant of the paired corpora per training run
	Epochs / cutoff	3 / 32,768 tokens
	Learning rate	10^{-5} , cosine, 5% warmup
RL	Rollouts per task	$N = 8$ (64 tasks per step)
	Schedule	103 steps
	Sampling	temperature 1.0, top- p 0.8, top- k 20
	Step limit / per-step cap	50 steps / 4,096 tokens (32B: 8,192)
	Context geometry	4,096 prompt + 34,816 response
	Learning rate	10^{-6}
	Clipping ($\epsilon_{lo}, \epsilon_{hi}$)	(0.2, 0.28)
	KL coefficient η	0.01; none on the terminal harness
	Prediction weight λ	10 against the token mean Z
		0.05 against a per-position mean on the terminal harness
	Reward coefficients (α, γ, κ)	(0.05, 0.0625, 0.02)
	Format term r_i^{fmt}	0.02 per step, clip ± 0.15 (repository); rate, weight 1.0 (terminal)
	Advantage aggregation	Token mean; Episode advantage unnormalized
	Group filtering	Uniform-outcome task groups dropped, not re-sampled
	Step-stream normalization	Per-task std with floor $\alpha + \gamma$
	Step-stream weight ω	0.5
	Step-term clip	± 2 after standardization
ANTE-TTO	Trigger	Abstention or predicted FAIL
	Candidates (K)	3 extra per trigger
	Scoring	PASS 3, NA 1, FAIL 0, +1 clean exit

F.3 Break-even derivation of the reward coefficients

Why the coefficients are derived quantities. Fixing the scale $\alpha > 0$, let q denote the probability that a trusted prediction is correct. On steps where a correct trusted forecast receives $+\alpha$, the expected score of trusting is $q\alpha - (1 - q)\gamma$, which is positive iff $q > \gamma/(\alpha + \gamma)$; choosing a target accuracy $q^* \in (0, 1)$ and

setting γ from it,

$$\gamma = \alpha \frac{q^*}{1 - q^*} \quad (\text{trusting is rewarded in expectation iff } q > q^*), \quad (21)$$

$$q^\circ = q^* - \frac{\kappa}{\alpha + \gamma} \quad (\text{indifference between trusting and abstaining, clean-exit forecasts}), \quad (22)$$

places the trust-versus-abstain break-even point q° strictly below the target, since abstaining costs the flat $\kappa > 0$ regardless of the prediction. A policy whose predictions are becoming reliable can therefore drop the corpus’s blanket abstention on epistemic steps before its accuracy reaches q^* , while trusting is rewarded in expectation only above q^* itself.

With the coefficients of Table 8, the target accuracy is q^* and the break-even point is q° . The task reward adds to the binary resolve outcome the bounded train-time format term of §3.3. The term credits executed steps with a well-formed belief block and penalizes unsolved rollouts near the response budget.

F.4 Compute and software stack

Training compute. We use NVIDIA A100 40GB GPUs for training: 64 GPUs at 7B, 14B, and 9B, and 128 GPUs at 32B.

Software stack. The repository-scaffold RL runs build on verl (Sheng et al., 2025) and the terminal runs use the open-instruct GRPO trainer released with Tmax (Iverson et al., 2026) on the Harbor. Rollout and evaluation inference use vLLM-based workers (Kwon et al., 2023). The repository scaffold is the official R2E-Gym tool interface (the base prompt of Appendix O).

G ANTE-TTO: candidate score, algorithm, and examples

G.1 Scoring rule and inference algorithm

Candidate score. ANTE-TTO scores a candidate by the belief it carries. For candidate k with predicted check field $\hat{y}^{\text{tst}(k)}$ and exit field $\hat{y}^{\text{exe}(k)}$, the score is

$$s(\beta^{(k)}) = \phi(\hat{y}^{\text{tst}(k)}) + \mathbf{1}[\hat{y}^{\text{exe}(k)} = 1], \quad \phi(\text{PASS}) = 3, \quad \phi(\text{NA}) = 1, \quad \phi(\text{FAIL}) = 0, \quad (23)$$

and the selected candidate is $\tilde{k} = \arg \max_{k \in \{0, 1, \dots, K\}} s(\beta^{(k)})$, with ties resolved toward $k = 0$, the action the policy generated first. The ordering $\text{PASS} \succ \text{NA} \succ \text{FAIL}$ ranks a predicted pass above an indeterminate outcome above a predicted failure; the exit term favors candidates the policy expects to execute cleanly. The rule consumes only the two value tokens of the belief, so it adds no model and no execution. Exactly one action is executed per step.

Algorithm 3 ANTE-TTO inference

Require: policy π_θ ; trigger set $\mathcal{E}$ (abstention or predicted FAIL); candidates per trigger K

Ensure: trajectory τ

```

1: for  $t = 1, 2, \dots$  until submission do
  // Phase 1: generate and parse the belief
2:  $u_t = (v_t, \beta_t, a_t) \leftarrow$  greedy decoding of  $\pi_\theta(\cdot \mid h_t)$ ;  $u^{(0)} \leftarrow u_t$ 
  // Phase 2: resample only where the belief signals uncertainty
3: if  $\beta_t \in \mathcal{E}$  then
4:   sample  $u^{(k)} = (v^{(k)}, \beta^{(k)}, a^{(k)}) \sim \pi_\theta(\cdot \mid h_t)$  for  $k = 1, \dots, K$ 
5:    $\tilde{k} \leftarrow \arg \max_{k \in \{0, \dots, K\}} s(\beta^{(k)})$ , ties toward  $k = 0$ 
6:    $u_t \leftarrow u^{(\tilde{k})}$ ;  $(v_t, \beta_t, a_t) \leftarrow u_t$ 
7: end if
  // Phase 3: execute the selected action
8:  $o_{t+1} \leftarrow \text{execute}(a_t)$ ;  $h_{t+1} \leftarrow (h_t, u_t, o_{t+1})$ 
9: end for
10: return  $\tau$ 
```

▷ Eq. 23

The trigger set and K take the values of Table 8. At a trigger step the draft stays greedy, and the three alternatives are sampled at temperature 1.0 under the same token cap and context, so an ANTE-TTO

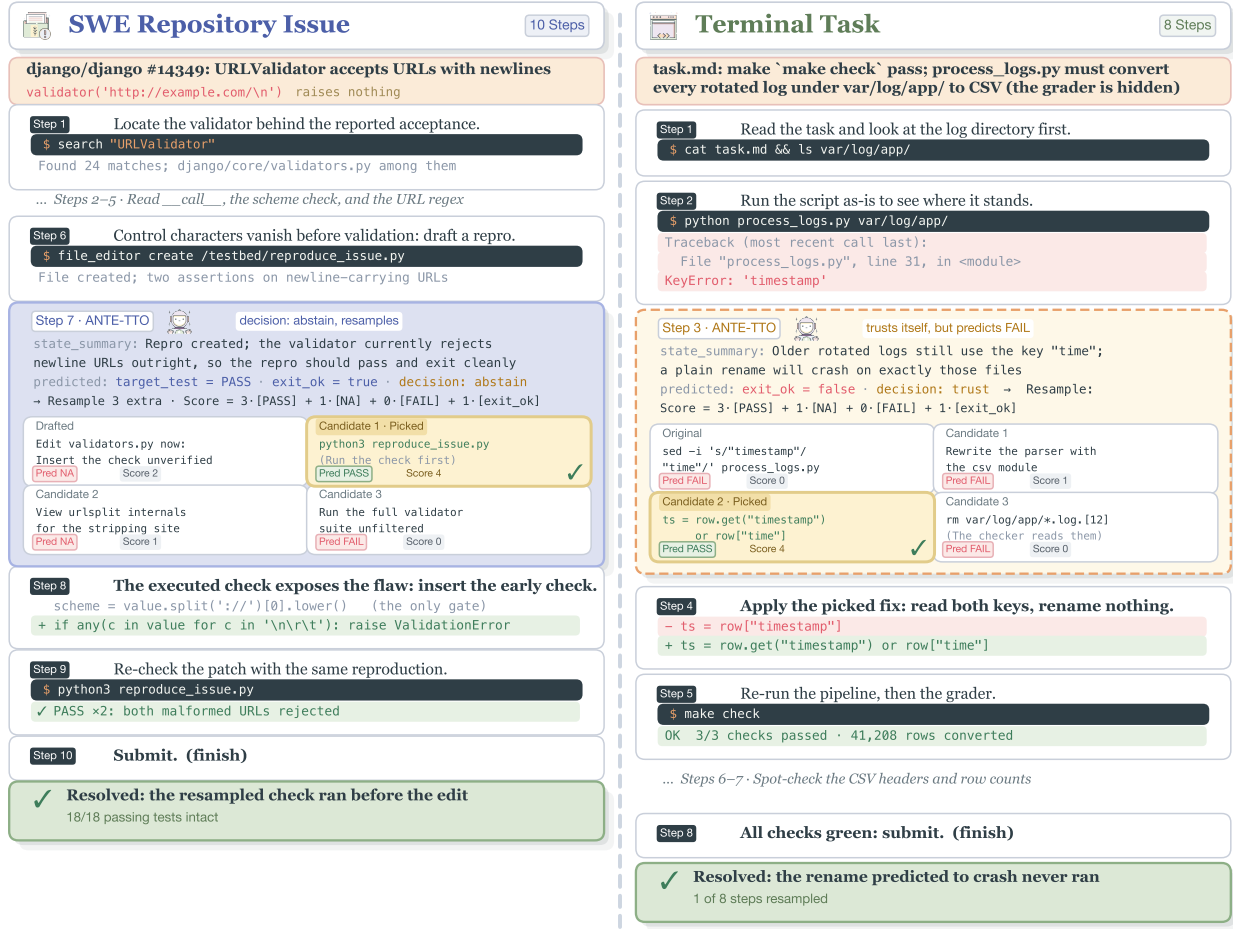


Figure 14 ANTE-TTO across the two harnesses. At a trigger step the policy resamples candidate actions and scores each by its own belief; the command predicted to crash never touches the environment.

trajectory mixes greedy and sampled steps. Token budgets count prompt and generated tokens, discarded candidates included, and are measured from the recorded call counts of the same evaluation runs. Figure 4 reports the marginal token cost of each setting and the averaged B_{tok} and nMTC.

G.2 Examples on different harnesses

On the repository scaffold (32B policy, `django__django-14349`), the step-7 abstention triggers resampling: the four candidates score $\{4, 2, 1, 0\}$ and the winner replaces the drafted command (full trace in Appendix N). On the terminal scaffold (Figure 14, right), the belief predicts a crash for the plain-rename fix, and the predicted failure triggers resampling. The read-both-keys fix wins with a score of 4, and the log-deleting shortcut scores zero.

H A worked example of BAPO

We compute the BAPO assignment (§3.3) on the configuration of Figure 3: one task, three rollouts, and reward coefficients $(\alpha, \gamma, \kappa) = (0.05, 0.0625, 0.02)$. We report the belief-relative and decision-relative advantages, which the assignment combines at weight $\omega = \frac{1}{2}$ after standardization. Conclusive steps are written as [prediction, decision] with the decision reward to the right; F and P stand for the predicted FAIL and PASS, a check mark means the prediction was correct, and a cross that it missed.

	Step t_a		Step t_b		Step t_c	
τ_1 ($R=1$)	[F, abstain]	-0.02	[F, trust ✓]	+0.05	[P, abstain]	-0.02
τ_2 ($R=0$)	[F, abstain]	-0.02	[P, trust ✗]	-0.0625	[P, abstain]	-0.02
τ_3 ($R=0$)	[F, trust ✓]	+0.05	[P, trust ✗]	-0.0625		

Group(F): $\hat{V}(F) = -0.0188$				Group(P): $\hat{V}(P) = -0.0463$			
abstain: $\hat{Q} = -0.0463$		trust: $\hat{Q} = +0.0088$		abstain: $\hat{Q} = -0.02$		trust: $\hat{Q} = -0.0725$	
$\hat{A} = \hat{Q} - \hat{V} = -0.0275$		$\hat{A} = \hat{Q} - \hat{V} = +0.0275$		$\hat{A} = \hat{Q} - \hat{V} = +0.0263$		$\hat{A} = \hat{Q} - \hat{V} = -0.0263$	
$r_{i,t}$	$\tau_1 t_a$	$\tau_2 t_a$	$\tau_1 t_b$	$\tau_3 t_a$	$\tau_1 t_c$	$\tau_2 t_c$	$\tau_3 t_b$
	-0.02	-0.02	+0.05	+0.05	-0.02	-0.02	-0.0625
$W_{i,t}$	+0.01	-0.1025	+0.03	-0.0125	-0.02	-0.02	-0.0825
$A_{i,t}^{\tilde{y}} = W - \hat{V}$	+0.029	-0.084	+0.049	+0.006	+0.026	+0.026	-0.036
$A_{i,t}^{\tilde{y},d} = W - \hat{Q}$	+0.056	-0.056	+0.021	-0.021	0	0	-0.010
$S_{i,t} = \frac{1}{2}(A_{i,t}^{\tilde{y}} + A_{i,t}^{\tilde{y},d})$	+0.043	-0.070	+0.035	-0.008	+0.013	+0.013	-0.023

A_i : $A_1 = +0.667$ $A_2 = -0.333$ $A_3 = -0.333$ Per-step advantage $A_{i,t} = A_i + \omega(\tilde{A}^{\tilde{y}} + \tilde{A}^{\tilde{y},d})$ after standardization
Forecast FAIL (F) Forecast PASS (P) tr = trust, ab = abstain; ✓ correct, ✗ missed forecast
 $r_{i,t}$: $+\alpha = +0.05$ Rightly trusted, $-\gamma = -0.0625$ Wrongly trusted, $-\kappa = -0.02$ Abstained; W Decision Return

Figure 15 BAPO worked example.

Episode advantages. $R = [1, 0, 0]$, mean $1/3$, so $A = [+0.667, -0.333, -0.333]$; every pragmatic step receives its trajectory’s A_i unchanged.

Decision return. W is the suffix sum of the decision rewards within each trajectory: for τ_1 , $W(t_c) = -0.02$, $W(t_b) = +0.03$, $W(t_a) = +0.01$; for τ_2 , $W(t_c) = -0.02$, $W(t_b) = -0.0825$, $W(t_a) = -0.1025$; for τ_3 , $W(t_b) = -0.0625$, $W(t_a) = -0.0125$.

Groups. Belief groups ignore step positions: $G(F) = \{\tau_1 t_a, \tau_1 t_b, \tau_2 t_a, \tau_3 t_a\}$ and $G(P) = \{\tau_1 t_c, \tau_2 t_b, \tau_2 t_c, \tau_3 t_b\}$. Rollout τ_1 contributes two members to $G(F)$, and $G(P)$ mixes steps from different positions. The group means are $\hat{V}(F) = -0.0188$ and $\hat{V}(P) = -0.0463$.

Belief-relative advantages. $A^{\tilde{y}} = W - \hat{V}$ rewards steps whose decision returns exceed their group mean. The trusted correct prediction $\tau_1 t_b$ gets the largest belief-relative advantage (+0.049); the opening step of the rollout with the lowest decision return ($\tau_2 t_a$) gets the most negative (-0.084). In this example the signs of the final advantages are still set by A_i : a correct trust inside a failed trajectory ($\tau_3 t_a$) remains negative overall.

Decision advantage. Splitting $G(P)$ by decision gives $\hat{Q}(P, \text{ABSTAIN}) = -0.02$ and $\hat{Q}(P, \text{TRUST}) = -0.0725$. The shared decision advantage $\hat{A}(P, d) = \hat{Q} - \hat{V}$ is therefore +0.0263 for abstain and -0.0263 for trust: under a predicted PASS, trusting carried negative expected reward in this batch. On $G(F)$ the sign flips, $\hat{A}(F, \text{TRUST}) = +0.0275$, because in this batch the FAIL forecasts were the ones that proved correct, so trusting them was rewarded. The decision is scored per belief, which is why the group key contains $\tilde{y}$. Before standardization, the step term $\frac{1}{2}(A^{\tilde{y}} + A^{\tilde{y},d})$ of the BAPO assignment is largest for the abstained FAIL forecast that opens the winning rollout ($\tau_1 t_a$, +0.043) and most negative for the same opening step of τ_2 ($\tau_2 t_a$, -0.07). The correct trust inside the failed rollout ($\tau_3 t_a$) keeps a positive belief-relative advantage ($A^{\tilde{y}} = +0.006$) even as its total advantage stays negative. The decomposition $A^{\tilde{y}} = \hat{A} + A^{\tilde{y},d}$ (Eq. 9) holds for every member listed above.

I Terminal instantiation and configuration differences

We redefine the outcome space, the anchor key, and the decision reward since the outcomes the terminal verifier can read are different. In the Harbor task format the hidden test runs after the episode, so the harness returns two signals. The exit status of each command is forecast by the exit field and settled by the next observation. The episode outcome is forecast by the check field and settled once by the verifier. Belief groups are keyed on the predicted exit field, the coordinate that recurs across rollouts, and the decision reward takes

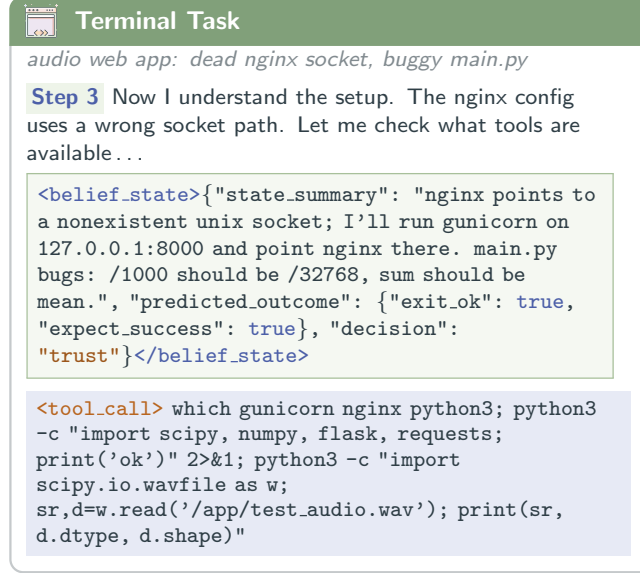


Figure 16 Belief state on the terminal harness.

Repository issue: per-step target check						
Step type χ	a_1 PRAG	a_2 EPI	a_3 PRAG	a_4 PRAG	a_5 EPI	a_6 PRAG
Check y_t^{tst}	NA	FAIL	NA	NA	PASS	NA
Exit y_t^{exe}	1	1	1	0	1	1
Supervised	✓	✓	✓	✓	✓	✓
Conclusive		✓			✓	
Reward r_t	0	$\sigma(\cdot)$	0	0	$\sigma(\cdot)$	0
Anchor key		$\hat{y}^{\text{tst}}$			$\hat{y}^{\text{tst}}$	

Terminal task: outcome as check field						
Step type χ	a_1 PRAG	a_2 EPI	a_3 PRAG	a_4 EPI	a_5 PRAG	a_6 PRAG
Check y_t^{tst}	y_i	y_i	y_i	y_i	y_i	y_i
Exit y_t^{exe}	1	1	0	1	1	1
Supervised	✓	✓	✓	✓	✓	✓
Anchored	✓	✓	✓	✓	✓	✓
Reward r_t	0	0	0	0	0	$\sigma(\cdot)$
Anchor key	$\hat{y}^{\text{exe}}$	$\hat{y}^{\text{exe}}$	$\hat{y}^{\text{exe}}$	$\hat{y}^{\text{exe}}$	$\hat{y}^{\text{exe}}$	$\hat{y}^{\text{exe}}$

Figure 17 Terminal instantiation. Left: repository issues. Right: terminal tasks.

the episode outcome. Eq. 4 is instantiated with the exit field as the supervised target at every step and a per-position denominator (Table 8).

Instantiation. The check field predicts the hidden verifier’s outcome, and the exit field predicts whether each command exits with code zero (Figure 16). The exit field is supervised on every step against the next observation’s exit code. The check field carries the episode outcome and enters credit through the decision reward rather than the prediction loss. The check coordinate of the realized outcome is constant within an episode, while the exit coordinate is read per step from the next observation. Credit is per-step as well: steps are anchored on their predicted exit field, and belief-decision groups split each belief group exactly as on repository issues. The hidden verifier evaluates the episode once, so the check field is scored after the last step (Table 7): $r_{i,t} = 0$ for $t < T_i$, $r_{i,T_i} = \sigma(d_{i,T_i}, \hat{y}_{i,T_i}, y_{i,T_i})$, and $W_{i,t} = r_{i,T_i}$. Here $\mathcal{C}_i := \{T_i\}$, and the belief groups are $G(\hat{y}) := \{(j, s) : m_{j,s} = 1, \hat{y}_{j,s}^{\text{exe}} = \hat{y}\}$ for $\hat{y} \in \{0, 1\}$. Steps of one rollout therefore share a return but not a baseline. The group of a step is set by its own predicted exit field, so a rollout’s steps are measured against different $\hat{V}$ and $\hat{Q}$ according to what each predicted. A step whose block does not parse is not anchored (Figure 17). At inference, the ANTE-TTO trigger of §3.4 reads the predicted exit field in place of the check field.

Terminal training configuration. The terminal variants train on the Tmax agent on Harbor as follows. The RL pool holds 3,558 tasks of the 14,601-task tmax-15k pool (Iverson et al., 2026). Each variant starts from its own fine-tuned checkpoint and uses its own prediction, format, and anchoring coefficients. Each episode has a 48K-token generation budget and a 120-step cap, and each training step uses 16 tasks with $N = 16$

rollouts each, 256 rollouts in all.

J Evaluation against concurrent methods on SWE-bench Verified

Table 9 places ANTE among open-weight methods on SWE-bench Verified at all three scales. At the two smaller scales, ANTE-7B reaches 27.2 against 24.1 for SWE-AGILE-8B (Lian et al., 2026), the best task-trained agent below 10B, and ANTE-14B reaches 46.2 against 30.1 for SWE-AGILE-14B, the best between 10B and 30B. Every entry is a pass@1 value, and external rows are as reported under each method’s own scaffold. The ANTE evaluation runs with a 64K context and a 200-step cap.

Table 9 Open-weight methods on SWE-bench Verified (pass@1, %) at all three scales. **Bold** marks the best and underline the second-best task-trained method per scale.

Method	Backbone	Training	Scaffold	Resolved (%)
<i><10B</i>				
Qwen2.5-Coder-7B	–	–	OpenHands	1.8
Qwen3-8B	–	–	ProRL Agent	9.6
SWE-Dev-7B	Qwen2.5-Coder-7B	SFT	OpenHands	23.4
R2E-Gym-7B	Qwen2.5-Coder-7B	SFT	R2E-Gym	19.0
Lingma-SWE-GPT-7B	Qwen2.5-Coder-7B	SFT	SWESynInfer	18.2
SWE-agent-LM-7B	Qwen2.5-Coder-7B	SFT	SWE-agent	15.2
SkyRL-Agent-7B-v0	Qwen2.5-Coder-7B	SFT+RL	OpenHands	14.6
SWE-Gym-7B	Qwen2.5-Coder-7B	SFT	OpenHands	10.6
SWE-AGILE-8B	Qwen3-8B	SFT+RL	R2E-Gym	<u>24.1</u>
ProRL Agent-8B	Qwen3-8B	RL	ProRL Agent	18.0
ANTE-7B	Qwen2.5-Coder-7B	SFT+RL	R2E-Gym	27.2
<i>10–30B</i>				
Qwen3-14B	–	–	ProRL Agent	15.4
Qwen2.5-Coder-14B	–	–	OpenHands	4.0
SWE-AGILE-14B	Qwen3-14B	SFT	R2E-Gym	<u>30.1</u>
R2E-Gym-14B	Qwen2.5-Coder-14B	SFT	R2E-Gym	26.8
ProRL Agent-14B	Qwen3-14B	RL	ProRL Agent	23.6
SkyRL-Agent-14B-v0	Qwen3-14B	RL	OpenHands	21.6
SWE-Gym-14B	Qwen2.5-Coder-14B	SFT	OpenHands	16.4
ANTE-14B	Qwen2.5-Coder-14B	SFT+RL	R2E-Gym	46.2
<i>>30B</i>				
Qwen3-32B	–	–	R2E-Gym	25.0
Qwen3-32B	–	–	ReAct	24.4
Qwen3-32B	–	–	OpenHands	23.2
Qwen2.5-Coder-32B	–	–	OpenHands	7.0
SWE-World-32B	Qwen2.5-Coder-32B	SFT+RL	R2E-Gym	<u>55.0</u>
FrogBoss-32B	Qwen3-32B	SFT	R2E-Gym	54.6
Meta-CWM-32B	Scratch	SFT+RL	CWM	53.9
SWE-Lego-32B	Qwen3-32B	SFT	OpenHands	52.6
DeepSWE-32B	Qwen3-32B	RL	R2E-Gym	42.2
SWE-agent-LM-32B	Qwen2.5-72B	SFT	SWE-agent	40.2
Skywork-SWE-32B	Qwen2.5-Coder-32B	SFT	OpenHands	38.0
OpenHands-LM-32B	Qwen2.5-Coder-32B	SFT	OpenHands	37.2
SWE-Dev-32B	Qwen2.5-Coder-32B	SFT	OpenHands	36.6
R2E-Gym-32B	Qwen2.5-Coder-32B	SFT	R2E-Gym	34.4
SWE-Gym-32B	Qwen2.5-Coder-32B	SFT	OpenHands	20.6
Kimi-Dev-72B	Qwen2.5-72B	SFT+RL	SWE-agent	48.6
SWE-Fixer-72B	Qwen2.5-72B	SFT	Retrieve-and-edit	32.8
Lingma-SWE-GPT-72B	Qwen2.5-72B	SFT	SWESynInfer	30.2
ANTE-32B	Qwen3-32B	SFT+RL	R2E-Gym	57.2

Sources for Table 9: SWE-AGILE (Lian et al., 2026), SWE-Dev (Wang et al., 2025a), R2E-Gym (Jain et al., 2025), Lingma SWE-GPT (Ma et al., 2024), SWE-agent-LM (Yang et al., 2025b), SWE-Gym (Pan et al., 2025), SkyRL-Agent (NovaSky Team, 2025), SWE-World (Sun et al., 2026), FrogBoss (Sonwane et al., 2025),

Meta-CWM (Meta FAIR CodeGen Team, 2025), Kimi-Dev (Yang et al., 2025c), DeepSWE (Luo et al., 2025), Skywork-SWE (Zeng et al., 2025), OpenHands LM (OpenHands Team, 2025), SWE-Fixer (Xie et al., 2025), ProRL Agent (Zhang et al., 2026), and SWE-Lego (Tao et al., 2026). Scaffolds: SWE-agent (Yang et al., 2024), OpenHands (Wang et al., 2024a), and ReAct (Yao et al., 2022). Terminal rows (Table 10): Nemotron-Terminal (Pi et al., 2026), TerminalTraj (Wu et al., 2026b), ECHO (Shrivastava et al., 2026); the Qwen3.5-9B base row with the Tmax agent on Harbor is our own measurement. The terminal base-model rows are from ECHO (Shrivastava et al., 2026) (Qwen3-8B and Qwen3-14B, five attempts at temperature 0.6) and TerminalTraj (Wu et al., 2026b) (the Qwen2.5-Coder-7B, 14B, and 32B readings, four runs); Nemotron-Terminal (Pi et al., 2026) reports 2.5 and 4.0 for the two Qwen3 bases without a trial count, and the OpenThinker-Agent model card (Open Thoughts Team, 2025) reports 0.0 for Qwen3-8B. The base-model scores on SWE-bench Verified are from Pan et al. (2025) (the zero-shot Qwen2.5-Coder-7B, 14B, and 32B readings), Zhang et al. (2026) (Qwen3-8B and 14B), and, for the three Qwen3-32B rows, Sonwane et al. (2025), Cao et al. (2025), and Tao et al. (2026). Table 9 is restricted to methods that report publicly released weights.

Variation across runs. Three-run standard errors are the $\pm$ entries of Tables 1 and 2.

K Terminal evaluation protocol

This section introduces the protocol behind Tables 2 and 10. External rows are as reported by each method under its own protocol and scaffold, and the run counts they state differ: TerminalTraj averages four runs (Wu et al., 2026b), Endless Terminals averages five (Gandhi et al., 2026), the published ECHO rows report pass@1 over five sampled attempts with Terminus-2 on Qwen3 backbones (Shrivastava et al., 2026), and the remaining rows state none. Our ECHO reproduction implements ECHO’s objective on the SkyRL trainer (Cao et al., 2025) with Qwen3.5-9B and our RL task pool, with 16 prompts and 16 samples per step, and is evaluated exactly like our own rows.

Table 10 places ANTE-9B among published open-weight terminal agents: it resolves 24.7% of Terminal-Bench 2.0, ahead of every task-trained agent below 20B in the table, including ECHO at 8B and 14B and our ECHO reproduction on the same backbone (21.4).

Table 10 Open-weight terminal agents on Terminal-Bench 2.0 (Resolved, %). **Bold** marks the best and underline the second-best task-trained method within each size band. †: our reproduction of ECHO (backbone Qwen3.5-9B, trained on our RL task pool).

Method	Backbone	Training	Scaffold	Resolved (%)
<i>Base models</i>				
Qwen2.5-Coder-7B	–	–	Terminus-2	0.0
Qwen2.5-Coder-14B	–	–	Terminus-2	1.2
Qwen2.5-Coder-32B	–	–	Terminus-2	4.5
Qwen3-8B	–	–	Terminus-2	1.6
Qwen3.5-9B	–	–	Tmax	16.9
Qwen3-14B	–	–	Terminus-2	4.3
<i>< 10B</i>				
TerminalTraj-7B	Qwen2.5-Coder-7B	SFT	Terminus-2	10.1
Nemotron-Terminal-8B	Qwen3-8B	SFT	Terminus-2	13.0
EndlessTerminals-8B (Gandhi et al., 2026)	Qwen3-8B	SFT+RL	Minimal loop	6.7
ECHO-8B	Qwen3-8B	RL	Terminus-2	5.2
OpenThinker-Agent-8B (Open Thoughts Team, 2025)	Qwen3-8B	SFT+RL	Terminus-2	4.9
ANTE-9B	Qwen3.5-9B	SFT+RL	Tmax	24.7
ECHO-9B†	Qwen3.5-9B	RL	Tmax	<u>21.4</u>
<i>10–30B</i>				
Nemotron-Terminal-14B	Qwen3-14B	SFT	Terminus-2	20.2
TerminalTraj-14B	Qwen2.5-Coder-14B	SFT	Terminus-2	<u>19.1</u>
ECHO-14B	Qwen3-14B	RL	Terminus-2	10.8

L Additional analyses of the belief under RL

We also analyze whether the belief protocol holds under RL and how well the emitted forecast judges the trajectory. We find that the protocol holds, and that the forecast judges the trajectory about as well as a dedicated reward model given the gold patch as a reference.

Belief protocol stability. At 32B under RL (Figure 18), the settled share of predictions holds, and so do the steps and response tokens per trajectory.

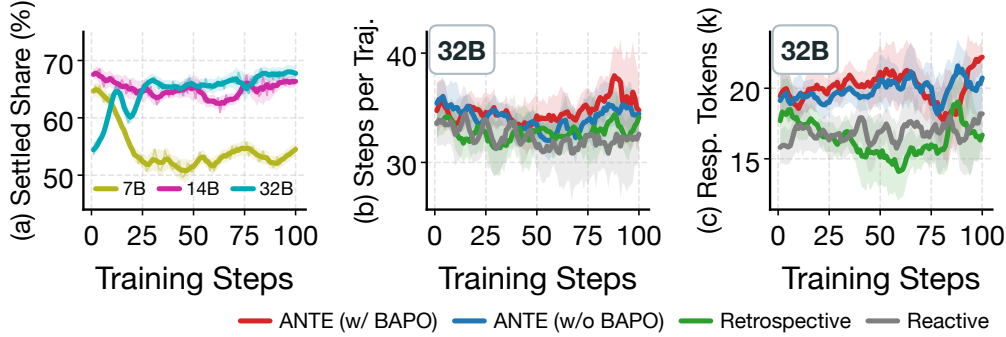


Figure 18 Belief protocol stability. (a) The settled share of predictions, the steps whose forecast is scored, for ANTE at 7B, 14B, and 32B; (b) steps and (c) response tokens per trajectory at 32B for the four variants.

Outcome judging against a reward model. A separately trained reward model is the standard tool for judging whether a trajectory resolves its task. We apply SWE-World’s 32B reward model (SW-RM) (Sun et al., 2026), with and without the gold patch, to ANTE-32B’s trajectories and compare its verdicts with the outcome the belief forecasts (Table 11). ANTE outperforms SW-RM and performs comparably even to SW-RM with the gold patch as reference.

Table 11 Trajectory-outcome judging. SW-RM: SWE-World’s 32B reward model, with or without the gold patch. Covered Acc. and MCC (Matthews correlation coefficient) count only judged trajectories, and Overall Acc. also counts the rest.

Judge	Covered Acc. (%)	MCC	Overall Acc. (%)
SW-RM (w/o gold)	65.5	+0.359	58.4
SW-RM (w. gold)	72.3	+0.454	63.2
ANTE	73.2	+0.482	73.2

M Failure modes without foresight

This section provides one episode for each failure mode of an agent that acts without a scored forecast (Figure 19).

Irreversible actions. An agent that tries to discard its own edit with `git checkout` finds the repository’s index corrupt and spends its last five steps repairing it (`matplotlib__matplotlib-22871`). The exit field puts a forecast on every command’s outcome, and a predicted clean run that crashes is a supervised miss (Eq. 4).

Silent regressions. On `django__django-11451` the patch passes every target test but breaks authentication for custom user models. The agent’s attempts to run the authentication tests fail, first on a missing module and then on the test settings, so it submits on its reproduction script, and only the final evaluation surfaces the damage. Fourteen further episodes of the same run end the same way. A per-step forecast, checked at every executed test, can surface such a regression at the next check.

No per-step credit. On `sympy__sympy-14531` one resolved trajectory adds a printer override at step 30, breaks two tests with it, and removes it at step 41. The single task reward covers the step that broke the build and the step that fixed it alike.

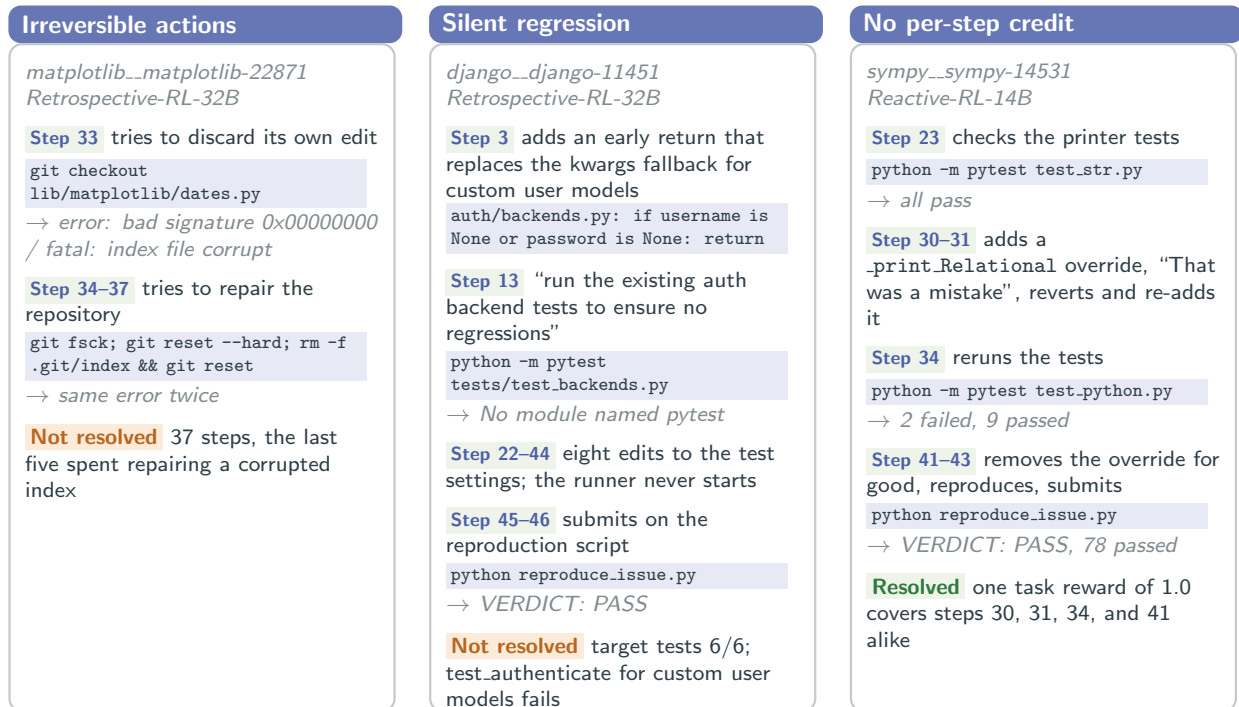


Figure 19 Failure modes without foresight.

N Case studies

We provide case studies from evaluation, training, and off-scaffold runs, each illustrating a different finding.

N.1 Foresight guides test-time optimization

On `django__django-14349` the policy abstains at steps 7 and 9. At step 7, ANTE-TTO samples three more candidate actions, scores all four by their beliefs, and executes the highest-scoring one instead of the original action. The executed check refutes its abstained forecast and exposes the issue. The run finishes with the task resolved and every previously passing test intact.

Steps 1–3 · locating the validator

```
<belief_state>{"state_summary": "The URL validation bug likely lives in the URLValidator class under django/core/validators.py.",
"predicted_outcome": {"target_test": "NA", "exit_ok": true}, "decision": "trust"}</belief_state>
```

```
<function=search> URLValidator
Found 24 matches; ./django/core/validators.py among them
```

```
<belief_state>{"state_summary": "core/validators.py holds the URL validation logic; its condensed structure locates the class.", "pre-
dicted_outcome": {"target_test": "NA", "exit_ok": true}, "decision": "trust"}</belief_state>
```

```
<function=file_editor> view django/core/validators.py
Condensed view of the file structure
```

```
<belief_state>{"state_summary": "URLValidator.__call__ likely mishandles a scheme or host case; locating its definition.", "pre-
dicted_outcome": {"target_test": "NA", "exit_ok": true}, "decision": "trust"}</belief_state>
```

```
<function=search> def __call__ (validators.py)
7 matches; the URLValidator's at line 101
```

Steps 4–6 · reading the validator, writing a script that triggers the bug

```
<belief_state>{"state_summary": "__call__ validates the scheme and then the full URL; inspecting the body around line 101.",
"predicted_outcome": {"target_test": "NA", "exit_ok": true}, "decision": "trust"}</belief_state>

<function=file_editor> view validators.py:101–124

Scheme check at 105; full-URL validation below

<belief_state>{"state_summary": "The URL regex at lines 86–92 defines the accepted structure; checking it against the reported case.",
"predicted_outcome": {"target_test": "NA", "exit_ok": true}, "decision": "trust"}</belief_state>

<function=file_editor> view validators.py:86–92

The compiled URL regex, scheme validated separately

<belief_state>{"state_summary": "Newer Python strips control characters before URL splitting, so URLs containing newlines pass
validation; writing a reproduction script for both reported cases.", "predicted_outcome": {"target_test": "NA", "exit_ok": true},
"decision": "trust"}</belief_state>

<function=file_editor> create /testbed/reproduce_issue.py

File created; two assertions on newline-carrying URLs
```

Step 7 of 10 · the winning resampled candidate

Let's run the reproduction script to confirm the issue:

```
<belief_state>{"state_summary": "Reproduction script created; the validator currently rejects URLs with newlines outright, so the
repro should pass and exit cleanly.",
"predicted_outcome": {"target_test": "PASS", "exit_ok": true}, "decision": "abstain"}</belief_state>

<function=execute_bash> python3 /testbed/reproduce_issue.py
```

Exit code: 0

✗ FAIL: Should have rejected URL with newline
✗ FAIL: Should have rejected URL with IPv6 containing newline

Step 8 · reads the failure as confirmation, drafts the fix

Great, we've confirmed the issue. [...] The control characters are being stripped by Python's URL parser before our validation code gets a chance to check them.

```
<belief_state>{"state_summary": "URLValidator accepts URLs with control characters because Python's URL parser strips them before
scheme validation runs; an early control-character check should fix this.", "predicted_outcome": {"target_test": "NA", "exit_ok": true},
"decision": "trust"}</belief_state>

<function=file_editor> str_replace validators.py: insert control-character check before scheme validation
```

Step 9 · re-checks its own patch

Now let's run our reproduction script again to verify the fix:

```
<belief_state>{"state_summary": "Control-character check inserted before scheme validation; the repro should now reject both mal-
formed URLs.", "predicted_outcome": {"target_test": "PASS", "exit_ok": true}, "decision": "abstain"}</belief_state>

<function=execute_bash> python3 /testbed/reproduce_issue.py
```

Exit code: 0

✓ PASS: Correctly rejected URL with newline
✓ PASS: Correctly rejected URL with IPv6 containing newline

Step 10 · submits

Perfect! Our fix works as expected. [...]

```
<belief_state>{"state_summary": "Fix verified: both malformed URLs are rejected and the reproduction passes cleanly.", "predicted_-
outcome": {"target_test": "NA", "exit_ok": true}, "decision": "trust"}</belief_state>

<function=finish> submit ⇒ RESOLVED, 1/1 target, 18/18 passing suite
```

N.2 Memory extends context but not foresight

This case shows that recording a failure is not the same as predicting one. On `psf__requests-1724`, a unicode method name crashes an upload deep inside the standard library, and the working fix coerces the method to a native string. What differs is whether and how each method corrects a wrong fix. Retrospective encodes the method to bytes, and its own memory then faithfully records the damage. Nothing in its objective asks the memory to look forward, so the record arrives only after the breakage exists; the run is still patching the repository around its own fix when its last command fails. ANTE carries the same information as a forecast. It keeps the fix on the native-string side, runs four consecutive abstained checks that separate pre-existing network failures from its own diff, and resolves in 27 steps. Reactive, with no carried state, concludes that its own tracing is flawed and submits without notice.

Retrospective · the memory records the damage after it exists

Step 26: “This is a Python 3 issue: [...] So my fix breaks py3 tests.” Step 33: “it failed with `TypeError` in `http.client` about bytes. That IS caused by my change.”

```
<memory_state>{"state_summary": "prepare_method currently shows the ORIGINAL code (not my modified version), meaning my earlier edits were lost - possibly git stash reverted them. Need to check git status and reapply."}</memory_state>
```

Step 50 fails on the bytes fix ⇒ **NOT RESOLVED** after 50 steps; its own memory at step 24 records the 43 tests the bytes change broke.

ANTE · the forecast separates its own diff from pre-existing failures (step 26 of 27)

“Without my change, 13 tests fail (all network/httpbin related). [...] The key point: the failures are all 503/httpbin network errors, pre-existing. My change didn’t introduce any new failures.”

```
<belief_state>{"state_summary": "Confirmed the 4 failing tests are pre-existing httpbin/network failures (503), not caused by my change; my fix coerces method to native str correctly.", "predicted_outcome": {"target_test": "PASS", "exit_ok": true}, "decision": "abstain"}</belief_state>
```

Exit code: 0 VERDICT: PASS

Step 27 trusts the verified picture and submits ⇒ **RESOLVED, 6/6 target, 56/56 passing suite.**

Reactive · no carried state; submits on a check it calls flawed (steps 43–44 of 44)

Step 43: “My tracing approach is flawed. Regardless, the core fix is clear [...]” Step 44: “This matches the actual upstream fix. I’ll submit.” `<function=finish>` submit ⇒ **NOT RESOLVED, 0/6 target, 45/56 passing suite.**

N.3 The belief sharpens as the world model trains

As shown in the case of `sphinx-doc__sphinx-8721`, before RL the forecast misses the executed check twice, and the task remains unresolved after 41 steps. After RL, the forecast matches the check, and the task is resolved in 26 steps.

Before RL (ANTE-SFT model) · step 12 of 41

Let’s run the reproduction script to confirm the issue:

```
<belief_state>{"state_summary": "The repro script triggers the reported bug but prints nothing, so the failure should manifest silently.", "predicted_outcome": {"target_test": "FAIL", "exit_ok": false}, "decision": "abstain"}</belief_state>
```

Exit code: 0

Test passed: No module pages were generated for EPUB as expected

Both fields miss: the run predicted a failing check with an unclean exit and the execution returned a clean pass. The same hypothesis and the same forecast repeat at step 16 and miss again; the run submits at step 41 with the task unresolved.

After RL · step 24 of 26

Let’s run our updated reproduction script:

```
<belief_state>{"state_summary": "The repro script exercises the buggy code path and should surface the reported error in the unfixed state.", "predicted_outcome": {"target_test": "FAIL", "exit_ok": false}, "decision": "abstain"}</belief_state>
```

```
Exit code: 1
UNEXPECTED: No module pages found in either format
```

The same forecast, now correct on both fields: the unfixed path does fail with an unclear exit. The agent patches `viewcode.py`, the step-25 rerun returns `SUCCESS` with a clean exit, and the run finishes resolved in 26 steps.

N.4 A confident forecast of failure is still a commitment

Trust is a decision about the prediction, not about the success of the step: the scoring rule returns $+\alpha$ for a trusted forecast that comes true with a clean exit, even when the outcome it forecast is a failing check. One such step appears in the training rollouts. Since the run exits cleanly and the check fails as forecast, the step is a conclusive hit scored $+\alpha$.

RL rollout, 32B · epistemic step 21: a trusted forecast of failure

```
<belief_state>{"state_summary": "Applied fix and removed debug; now re-running the reproduction script.", "predicted_outcome":
{"target_test": "FAIL", "exit_ok": true}, "decision": "trust"}</belief_state>

<function=execute_bash> cd /testbed && python reproduce_issue.py
```

```
Exit code: 0 VERDICT: FAIL
```

N.5 Foresight transfers beyond agentic coding

Protocol of Table 3. Every method applies the benchmark’s standard chain-of-thought prompt to obtain its answers; a generation with no extractable answer counts as wrong. LCB-exec is the execution split of LiveCodeBench; BBH-hard and BBH-state combine official BIG-Bench Hard task sets; MMLU-Pro covers the physics, chemistry, and biology categories. Problem counts: CruxEval-O 800, CruxEval-I 800, LCB-exec 479, BBH-hard 1,000, BBH-state 1,250, MATH500 500, AIME’25 30, AMC23 40, Minerva 272, MMLU-Pro 600, SciBench 692.

In this case, ANTE spontaneously predicts and then traces the function to check; the Reactive control simulates once and stops. The foresight ability makes ANTE’s reasoning more reliable.

ANTE 32B · CruxEval-I sample_423: construct, then trace the actual function

```
“So the input could be [4, 2, 5, 1, 3, 2, x] where  $x \neq 4$ . [...] Hmm, but let me verify by tracing the actual function.”

i=6: selfie[6]=x. If  $x==4$ , remove last.  $x \neq 4$  so no removal.
i=5: selfie[5]=2  $\neq 4$  [...] i=0: selfie[0]=4==4, remove last  $\rightarrow$  [4, 2, 5, 1, 3, 2]

“That works! But wait, [...] Let me verify with  $x=1$ .” [the full trace repeats and passes] “Let me verify with  $x=9$ : [...]
same result. Correct.”  $\Rightarrow$  answer [4, 2, 5, 1, 3, 2, 7], graded correct.
```

Reactive 32B · the same item: one mis-simulated branch, no re-check

```
“Consider original = [4, 2, 5, 1, 3, 2, 4]. [...] i=0: 4==4  $\rightarrow$  remove selfie[6]=2.” [the function removes the last element; the
trace removes the seventh of eight and never re-runs] “My answer [4, 2, 5, 1, 3, 2, 4, 2] is correct.”  $\Rightarrow$  graded wrong.
```

O Instruction templates

We list the prompts used in this paper, from corpus construction through fine-tuning, RL, and evaluation. Table 12 gives a brief overview of the prompt categories.

O.1 Repository templates

Table 12 Prompt contents. Green-framed prompts carry the belief protocol; blue-framed prompts show the shared scaffold.

Prompt	Stage	Applies to
Repository-issue base system prompt	SFT, RL, evaluation	R2E-Gym scaffold
Belief protocol (repository issues)	SFT, RL, evaluation	ANTE variants
Memory protocol (Retrospective)	SFT, RL, evaluation	Retrospective
Belief protocol (terminal tasks)	SFT, RL, evaluation	ANTE on terminal tasks
Collection-time belief protocol	Corpus construction	The trajectory collector

Repository-issue base system prompt (R2E-Gym)

You are a programming agent who is provided a github issue and repository bash environment and is tasked to solve
→ certain tasks (e.g., file localization, testcase generation, code repair and editing etc) to resolve the issue.

We have access to the following functions:

— BEGIN FUNCTION #1: file_editor —

Description:

Custom editing tool for viewing, creating and editing files

- State is persistent across command calls and discussions with the user
- If path is a file, view displays the result of applying cat -n. If path is a directory, view lists non-hidden files and
→ directories up to 2 levels deep
- The create command cannot be used if the specified path already exists as a file
- If a command generates a long output, it will be truncated and marked with <response clipped>
- The undo_edit command will revert the last edit made to the file at path

Notes for using the str_replace command:

- The old_str parameter should match EXACTLY one or more consecutive lines from the original file. Be mindful of
→ whitespaces!
- If the old_str parameter is not unique in the file, the replacement will not be performed. Make sure to include
→ enough context in old_str to make it unique
- The new_str parameter should contain the edited lines that should replace the old_str

Parameters: // ... omitted: unmodified R2E-Gym tool schema ...

— END FUNCTION #1 —

— BEGIN FUNCTION #2: execute_bash —

Description:

Execute a bash command in the terminal.

Behavior notes:

- If a command may run indefinitely (long-running), consider running it in the background and redirecting output, e.g.
→ python3 app.py > server.log 2>&1 &.
- If the bash command returns exit code -1, it means the process is still running. The assistant may:
- Call this function again with command as an empty string ("") to retrieve additional logs.
- Send more input to STDIN of the running process by calling this function again with command set to the text input.
- Send command="ctrl+c" to interrupt the currently running process.
- If the command times out, it will be interrupted (SIGINT). The assistant may then retry or do further steps if
→ needed.

Parameters: // ... omitted: unmodified R2E-Gym tool schema ...

— END FUNCTION #2 —

— BEGIN FUNCTION #3: search —

Description:

Search for a term in a directory or a single file.

- If path is a directory (or unspecified, default is .), it recursively searches all non-hidden files and directories for the
→ search term.
- If path points to a file, it runs a grep -n in that file to show line numbers matching the search term.
- If more than 100 files match in a directory search, results are truncated and the tool will inform you to narrow your
→ search.
- If no matches are found, it will inform you as well.

Parameters: // ... omitted: unmodified R2E-Gym tool schema ...

Repository-issue base system prompt (R2E-Gym) (continued)

— END FUNCTION #3 —

— BEGIN FUNCTION #4: finish —

Description:

Finish the interaction once the task is complete or if no further progress can be made.

Behavior notes:

- The submit command finalizes your output.

Parameters:

1. command (string, required)

Currently allowed value: [submit]

2. result (string, optional)

The result text or final message to submit. Defaults to an empty string if not provided.

— END FUNCTION #4 —

If you choose to call a function ONLY reply in the following format with NO suffix:

<function=example_function_name>

<parameter=example_parameter_1>value_1</parameter>

<parameter=example_parameter_2>

This is the value for the second parameter

that can span

multiple lines

</parameter>

</function>

<IMPORTANT>

Reminder:

- Function calls MUST follow the specified format, start with <function= and end with </function>
- Required parameters MUST be specified
- Only call one function at a time
- VERY IMPORTANT: Each response must include both reasoning (as natural text) and function call (in above format) $\hookrightarrow$ to solve the task.

The protocol's two step types, EPISTEMIC and PRAGMATIC, are the operational form of the action-type map χ , and its machine-checkable condition is the conclusive-step condition of the scoring rule.

Belief protocol (repository issues)

--- ADDITIONAL OUTPUT PROTOCOL: belief-state block ---

EVERY response MUST have exactly this structure, in this order:

PART 1 — <belief_state>: Begin with your reasoning as plain text (no special tags), then emit exactly one belief-state

$\hookrightarrow$ block:

<belief_state>

{

"state_summary": "<ONE sentence, PAST+PRESENT only: what you have established so far and what is currently $\hookrightarrow$ broken. This is your running memory across steps. Never state an expected or intended outcome as an accomplished

$\hookrightarrow$ fact — if this step's action has not executed yet, its result does not belong here.>",

"predicted_outcome": {

"target_test": "<PASS | FAIL | NA — your prediction of the checking outcome of THIS action; NA only if this $\hookrightarrow$ action is not a check or its outcome will not be machine-readable>",

"exit_ok": <true | false — will the command itself run without crashing?>

},

"decision": "<trust | abstain>"

}

</belief_state>

PART 2 — exactly ONE function call, and NOTHING after it.

Belief protocol (repository issues) (continued)

Rules:

1. Never use fixed default values; predict each field from your actual reasoning about the current state. Predict `exit_ok` $\hookrightarrow$ as your call on whether the command will run cleanly — do NOT default to true; if you are genuinely unsure, keep your prediction and mark the step "abstain". Wrong predictions are expected and acceptable.

2. "decision" — your confidence in the predicted_outcome above, on EVERY step:
 - "trust": you stand by this prediction — you would act on it without needing to read the real outcome first. This includes a confident prediction of failure (e.g. `target_test` "FAIL" or `exit_ok` false that you are sure of).
 - "abstain": you are not sure; you will read the real outcome before relying on it.The decision is scored ONLY where the prediction carries a machine-checkable verdict (epistemic steps): a trusted prediction that is fully correct with a clean exit is rewarded, a trusted prediction with any error is penalized, and abstaining carries a small flat cost.

On a pragmatic step, your `exit_ok` is scored as a prediction.

It reflects how sure you are; both choices are legitimate on any step.

STEP TYPES (they decide what `target_test` means):

- EPISTEMIC step (a check): runs a test or reproduce script (`pytest / tox / unittest / python <name>_test.py / python reproduce_<name>.py`). Its output can prove a verdict.
- PRAGMATIC step: everything else (viewing, searching, editing, installing, submitting). A pragmatic step never yields a test verdict, so its `target_test` is always "NA" and its only substantive prediction is `exit_ok`.

`target_test` ON AN EPISTEMIC STEP

- "PASS": the run will cleanly show the target behavior working.
- "FAIL": the run will cleanly show the bug still present.
- "NA": the output will not support a clear verdict (e.g. a broad suite run with unrelated noise). Predict "NA" only when that is genuinely your expectation — if a clear verdict was in fact derivable, "NA" counts as a wrong prediction.

Your `predicted_outcome` is scored against the real execution on every step, whatever your decision is.

3. `state_summary` is persistent memory, not a plan: write what you KNOW (you have NOT executed this step's action yet).
4. When you write a reproduce/verification script, it MUST decide the outcome INSIDE the script and print a single final line "VERDICT: PASS" or "VERDICT: FAIL" — so `target_test` is machine-checkable.
5. VERIFICATION IS MANDATORY BEFORE FINISHING: after your final edit, you MUST re-run the reproduction or relevant tests and confirm the fix before calling finish. Never call finish on an unverified change.
6. Do not rush: explore the repository and verify intermediate steps as needed. Only finish when you are confident. Depth is preferred over guessing; being comprehensive is fine.
7. Output exactly one function call per response. Never write anything after the closing `</function>` tag.
8. Your reproduce/verification script MUST contain its own timeout logic (about 20 seconds): if it runs longer, it should print a timeout message so the outcome stays machine-readable.
9. If a `file_editor` edit fails, view the file near the edit location before retrying. Never repeat the same failing action; if something failed twice, change your approach.
10. `file_editor` `old_str/new_str` must NOT include line numbers — line numbers appear only in views.
11. The final submission step also carries the full belief-state block.
12. Never wrap your reasoning in thinking tags or any other markup; plain text only.

The memory protocol keeps the running summary and removes every forward-looking field.

Memory protocol (Retrospective)

--- ADDITIONAL OUTPUT PROTOCOL: memory-state block ---

EVERY response MUST have exactly this structure, in this order:

PART 1 — `<memory_state>`: Begin with your reasoning as plain text (no special tags), then emit exactly one memory-state block:

`<memory_state>`

{

"state_summary": "<ONE sentence, PAST+PRESENT only: what you have established so far and what is currently broken. This is your running memory across steps. Never state an expected or intended outcome as an accomplished

Memory protocol (Retrospective) (continued)

```
↪ fact — if this step's action has not executed yet, its result does not belong here.>"
}
```

```
</memory_state>
```

PART 2 — exactly ONE function call, and NOTHING after it.

Rules:

1. `state_summary` is persistent memory, not a plan: write what you KNOW (you have NOT executed this step's action yet).
2. When you write a reproduce/verification script, it MUST decide the outcome INSIDE the script and print a single final line "VERDICT: PASS" or "VERDICT: FAIL" — so the outcome is machine-checkable.
3. VERIFICATION IS MANDATORY BEFORE FINISHING: after your final edit, you MUST re-run the reproduction or relevant tests and confirm the fix before calling finish. Never call finish on an unverified change.
4. Do not rush: explore the repository and verify intermediate steps as needed. Only finish when you are confident. Depth is preferred over guessing; being comprehensive is fine.
5. Output exactly one function call per response. Never write anything after the closing `</function>` tag.
6. Your reproduce/verification script MUST contain its own timeout logic (about 20 seconds): if it runs longer, it should print a timeout message so the outcome stays machine-readable.
7. If a `file_editor` edit fails, view the file near the edit location before retrying. Never repeat the same failing action; if something failed twice, change your approach.
8. `file_editor old_str/new_str` must NOT include line numbers — line numbers appear only in views.
9. Never wrap your reasoning in thinking tags or any other markup; plain text only.

O.2 Terminal and collection templates

The terminal-harness variant carries the block through the (χ, ψ) interface (Appendix C.5) with two prediction fields: the per-step `exit_ok` and the episode-level `expect_success`.

Belief protocol (terminal tasks)

In EVERY response, immediately after your reasoning and BEFORE the THOUGHT line, emit exactly one belief-state
↪ block:

```
<belief_state>{"state_summary": "...", "predicted_outcome": {"exit_ok": true, "expect_success": true}, "decision": "  
↪ trust|abstain"}</belief_state>
```

Field rules:

1. "state_summary": ONE sentence, PAST+PRESENT only: what you have established so far and what is currently
↪ broken. MUST be concrete and specific to THIS task: name the actual files, commands, services, or config keys
↪ involved. Generic filler is not acceptable, even on easy or final steps.
2. "predicted_outcome": predict BEFORE seeing any output:
 - "exit_ok": true if you expect this command to exit with code 0, false otherwise. Always predict this, for every
↪ command.
 - "expect_success": your judgment of the FINAL outcome: if you continue your current approach to completion, will
↪ the hidden task verifier ultimately PASS? An unfinished task is NOT a reason for false; discovering your approach is
↪ wrong, or being stuck, IS. Update it every step as evidence arrives; do not default to true.
3. "decision": your confidence in the predicted_outcome above, on EVERY step: "trust" = you stand by this prediction (
↪ including a confident prediction of failure); "abstain" = you are unsure and the real outcome must decide.

The belief-state block is mandatory in every step, including the final submission step, where "expect_success" carries
↪ your final self-assessment: will the hidden task verifier PASS?
Values must be valid JSON on a single line.

During collection the agent writes its belief at the head of each response. During curation, we place the block in the post-reasoning position of the training protocol, re-derive the `predicted_outcome` fields from the logged observations with the verifier ψ (§3.2), and replace the decision by the fixed-rule label of Appendix D.

Collection-time belief protocol (corpus construction)

--- ADDITIONAL OUTPUT PROTOCOL: belief-state block ---

At the START of EVERY response, before any other text, emit exactly one belief-state block:

```
<belief_state>
{
  "state_summary": "<one sentence, past+present: what you have established so far and what is currently broken>",
  "predicted_outcome": {
    "target_test": "<PASS | FAIL | NA — your prediction of the checking outcome of THIS action; NA if this action is
↳ not a check or its outcome will not be machine-readable>",
    "exit_ok": "<true | false — will the command itself run without crashing?>"
  },
  "decision": "<trust | abstain — your confidence in the predicted_outcome above, on EVERY step, regardless of step
↳ type or predicted value: trust = you stand by this prediction (including a confident prediction of failure); abstain =
↳ you are unsure and will read the real outcome before relying on it. Never derive this from the action type or from
↳ exit_ok's value; it reflects only how sure you are.>"
}
</belief_state>
```

Rules:

1. Never use fixed default values; predict each field from your actual reasoning about the current state.
2. decision vocabulary is exactly: trust, abstain.
3. state_summary is persistent memory, not a plan: write what you KNOW (you have NOT executed this step's action
↳ yet).
4. When you write a reproduce/verification script, it MUST decide the outcome INSIDE the script (compare actual vs expected) and end by printing exactly one explicit verdict line:
print("VERDICT: PASS") or print("VERDICT: FAIL"). Never end such a script with a bare value dump (e.g. print(result) alone) — an outcome the verifier cannot read wastes the run.
Print VERDICT only at that single final decision point, never speculatively.

After the belief-state block, continue with your normal reasoning and exactly one function call.