CO-EVOLVING ACTING AND FORECASTING FOR LANGUAGE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training language agents is bottlenecked by the high cost of long-horizon rollouts, yet standard agentic reinforcement learning (RL) learns only from a terminal scalar reward, leaving the supervision in intermediate steps largely unused. Prior work exploits this supervision by predicting next observations, either with a separate world model or as an auxiliary loss on the policy. These objectives, however, are ill-suited to agentic settings: (1) observations such as HTML pages and terminal logs are long and noisy, and (2) single-step prediction mismatches the long-horizon RL objective. We propose COPE, a simple and effective method that adds multi-step action forecasting as an auxiliary loss. On successful on-policy trajectories, COPE trains the policy backbone on the action plan: the current action and K future actions predicted from the earlier history, without the intervening observations. Working in the action space sidesteps environmental noise, and forecasting multiple steps ahead better matches the horizon of policy optimization. We show that this loss optimizes success-conditioned action marginals, so acting and forecasting can be learned jointly within a single shared backbone. Across five benchmarks (ALFWorld, ScienceWorld, WebShop, τ^2-bench, and AppWorld), COPE (1) outperforms GRPO and observation-prediction baselines on aggregate metrics in simulated environments and improves the overall metrics on two realistic applications despite a negative Airline result, (2) is up to $1.9\times$ and $2.0\times$ more sample-efficient than ECHO and GRPO, respectively, and (3) needs neither a separate model nor a pre-training stage. A separated-forecaster analysis further supports implicit temporal ensembling through the shared backbone.

1 INTRODUCTION

Large language models (LLMs) enable autonomous agents to tackle complex, long-horizon tasks such as web navigation (Zheng et al., 2024; Deng et al., 2023; Zhang et al., 2025), agentic coding (Yang et al., 2024; Hong et al., 2023; Sapkota et al., 2025), and scientific discovery (Zheng et al., 2025; Buehler, 2024; Lu et al., 2024; Yamada et al., 2025). However, training such agents remains expensive: each rollout involves many interaction steps and tool calls, so extracting as much learning signal as possible from a limited number of interactions is crucial (Liu & van der Schaar, 2025). Yet standard agentic reinforcement learning (RL) learns only from a scalar reward at the end of each trajectory, leaving the supervision contained in intermediate steps largely unused.

To exploit more intermediate supervision, prior work has turned to environment feedback returned at each step. One line of work trains a separate LLM as the world model from environment feedback and uses it to plan or to synthesize additional experience (Chen et al., 2025; Ding et al., 2026). These methods require training and serving an additional model, and a simulator learned from limited trajectories is prone to overfitting and compounding errors. Another line of work uses a single LLM backbone and treats environment feedback as extra supervision for the policy itself. These methods add an auxiliary next-observation prediction loss to policy optimization (Shrivastava et al., 2026; Lu et al., 2026), learn to predict the next state before or alongside policy optimization (Zhang et al., 2026; Yu et al., 2026), or additional step-wise reward turns whose observations shift the policy's next action (Kong et al., 2026). Despite their differences, these methods share two design choices: their auxiliary objective is anchored on observations, and it is defined over a single step, i.e., the transition from the current step to the next.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

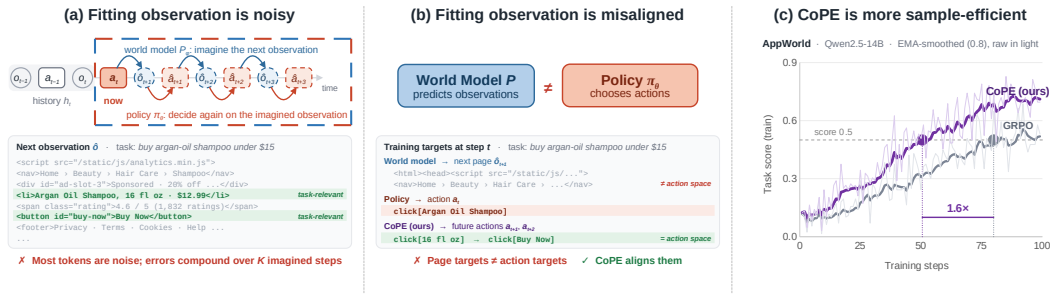

Figure 1: **Motivation of CoPE for sample-efficient RL.** (**a**) A web observation contains many task-irrelevant tokens, whereas the corresponding action is short and decision-relevant. (**b**) Observation-prediction methods target the next page $\hat{o}_{t+1}$, whereas CoPE forecasts the next K actions; the illustration uses the default $K = 2$. (**c**) On AppWorld, CoPE reaches a training task score of 0.5 with $1.6\times$ fewer RL steps than GRPO.

Key challenges. Both choices limit their effectiveness in complex agentic environments. **(1)** Environmental noise. In web navigation or agentic coding, observations are long, unstructured, and cluttered with irrelevant content (e.g., HTML DOM trees or verbose terminal logs). A model trained to predict them spends much of its capacity on tokens that have little bearing on decision-making. **(2)** Horizon mismatch. RL optimizes the policy for long-horizon success, whereas single-step objectives supervise only step-wise transitions. When both share one backbone, the auxiliary gradients can push the model to fit local dynamics rather than what multi-step policy requires, and can even conflict with policy optimization.

Our solution. To solve these two challenges, we propose CoPE, a simple and effective method that replaces single-step observation fitting with *multi-step action forecasting*. On successful on-policy trajectories, CoPE trains the policy backbone to generate $a_{t:t+K}$ from h_t: a success-filtered current-action target at $k = 0$ together with K future-action targets that skip the intervening observations. This design addresses both challenges: **(1)** Operating in the action space sidesteps environmental noise: the model never reconstructs web pages or terminal logs and learns only what matters for acting; **(2)** Forecasting multiple steps ahead better matches the horizon of policy optimization: instead of modeling a single-step transition, the model learns where a successful trajectory is heading. Formally, we show that the auxiliary loss optimizes success-conditioned action marginals, so acting and forecasting can be learned jointly within a single shared backbone.

Main findings. We evaluate CoPE on five benchmarks spanning three simulated environments (ALFWorld (Shridhar et al., 2020), ScienceWorld (Wang et al., 2022), and WebShop (Yao et al., 2022)) and two realistic applications (τ^2-bench (Barres et al., 2025) and AppWorld (Trivedi et al., 2024)). Our experiments yield three main findings. **(1)** Broad improvements: CoPE outperforms GRPO (Shao et al., 2024) and observation-prediction baselines (ECHO (Shrivastava et al., 2026) and RWML (Yu et al., 2026)) on aggregate metrics in the three simulated environments, while improving the overall metrics on both realistic applications despite a negative result on the Airline domain of τ^2-bench. **(2)** Sample efficiency: On ScienceWorld, CoPE is up to $1.9\times$ and $2.0\times$ more sample-efficient than ECHO and GRPO, respectively. **(3)** Low overhead: On ScienceWorld, the forecasting pass adds 11% time per training step on average, while shorter rollouts reduce the measured end-to-end step time by 9%; CoPE requires neither a separate model nor a pretraining stage.

2 Related Works

Sample-efficient agentic RL. Methods for sample-efficient agentic RL follow two main directions. The first makes better use of outcome rewards, either by refining credit assignment within trajectories (Feng et al., 2025; Wang et al., 2025; Luo et al., 2025; Wang et al., 2026b; Kong et al., 2026) or by converting completed interactions into extra policy supervision via hindsight rewriting or on-policy distillation (Hu et al., 2025; Yang et al., 2026; Wu et al., 2026; Zheng et al., 2026). These methods change how sparse outcomes are assigned or reused, without necessarily introducing a predictive objective. The second learns from the environment feedback available at every step, typically by adding auxiliary prediction of future states or observations to policy training (Zhang et al., 2026; Shrivastava et al., 2026; Lu et al., 2026). Such objectives provide dense token-level supervision but

generally focus on a single environment transition. COPE combines elements of both directions: success filtering reuses outcome rewards, while multi-horizon prediction learns from how trajectories unfold in action space rather than reconstructing observations.

Multi-horizon prediction. COPE is closely related to several forms of multi-horizon and outcome-conditioned prediction. Multi-token prediction trains language models to predict several future tokens (Gloeckle et al., 2024), while robotic policies such as ACT and Diffusion Policy predict sequences of future actions (Zhao et al., 2023; Chi et al., 2023). Multi-token prediction is closest architecturally because multiple offsets are learned by one backbone, whereas action chunking is closest in the structure of its targets. Outcome-conditioned sequence modeling conditions policies on desired returns, while self-imitation emphasizes high-return trajectories (Chen et al., 2021; Oh et al., 2018); both connect policy learning to trajectory-level outcomes. Accordingly, the $k = 0$ term in COPE is success-filtered imitation, while the $k \geq 1$ terms extend this supervision to multiple future action offsets. COPE uses these predictions as an auxiliary objective during on-policy GRPO: targets are regenerated from successful rollouts, and the predicted future actions are not executed as a chunk but affect acting through the shared backbone.

3 PRELIMINARIES

Language agents. We consider a long-horizon task in which an LLM policy π_θ interacts with an environment for T steps. At step t, the agent holds the history $h_t = (o_{\leq t}, a_{<t})$, samples an action $a_t \sim \pi_\theta(\cdot \mid h_t)$, and receives an observation $o_{t+1} \sim P(\cdot \mid h_t, a_t)$ from the unknown environment transition P, so that $h_{t+1} = (h_t, a_t, o_{t+1})$. This produces a trajectory $\tau = (o_0, a_0, o_1, \ldots, a_{T-1}, o_T)$. We focus on sparse-reward settings, where the agent receives only a terminal success signal $Y(\tau) \in \{0, 1\}$, and policy learning maximizes $J(\theta) = \mathbb{E}_{\tau \sim (\pi_\theta, P)}\big[Y(\tau)\big]$.

Policy optimization. A common choice for optimizing $J(\theta)$ is Group Relative Policy Optimization (GRPO) (Shao et al., 2024), which avoids training a critic. For each task, GRPO samples a group of G trajectories $\{\tau_i\}_{i=1}^{G}$ from the rollout policy π_{old} and assigns each the normalized advantage $\hat{A}_i = (Y(\tau_i) - \mu)/(\sigma + \delta)$, where μ and σ are the mean and standard deviation of rewards in the group. With the per-step ratio $\rho_{i,t}(\theta) = \pi_\theta(a_{i,t} \mid h_{i,t})/\pi_{\text{old}}(a_{i,t} \mid h_{i,t})$, GRPO minimizes

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{G}\sum_{i=1}^{G}\frac{1}{T_i}\sum_{t=0}^{T_i-1}\min\Big(\rho_{i,t}(\theta)\hat{A}_i,\ \text{clip}\big(\rho_{i,t}(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_i\Big). \tag{1}$$

Because the reward evaluates an entire trajectory, every action in τ_i receives the same advantage $\hat{A}_i$. The learning signal thus says only whether a trajectory succeeded, not how it unfolded: the intermediate steps, which record how each decision led to the next, provide no supervision of their own. This motivates extracting additional supervision from the temporal structure of the collected trajectories.

4 FROM OBSERVATION FITTING TO ACTION FORECASTING

Classical world model. The most common way to extract such supervision is to model the environment. A classical world model learns a parametric approximation $P_\phi \approx P$ by fitting the observed transitions:

$$\mathcal{L}_{\text{WM}}(\phi) = -\,\mathbb{E}_{(h_t, a_t, o_{t+1})}\big[\log P_\phi(o_{t+1} \mid h_t, a_t)\big]. \tag{2}$$

This objective is used either to train a separate model (Lin et al., 2023; Chae et al., 2024) or as an auxiliary loss on the policy backbone with $\phi = \theta$ (Shrivastava et al., 2026; Lu et al., 2026). It exhibits both challenges in Sec. 1. First, every token of o_{t+1} is a target, although most are irrelevant to the task (*environmental noise*). Second, it is a single-step objective: its target follows directly from the observed pair (h_t, a_t), whereas the policy is optimized for multi-step success (*horizon mismatch*).

Future-action marginal. To match the multi-step nature of RL, we instead predict what the agent will do k steps ahead, directly from h_t. The intermediate actions and observations are not yet available at step t, so this prediction must marginalize over them. For a policy π, rolling out k steps from h_t produces a segment $\tau_{t:t+k} = (a_t, o_{t+1}, \ldots, a_{t+k-1}, o_{t+k})$, and marginalizing over this segment

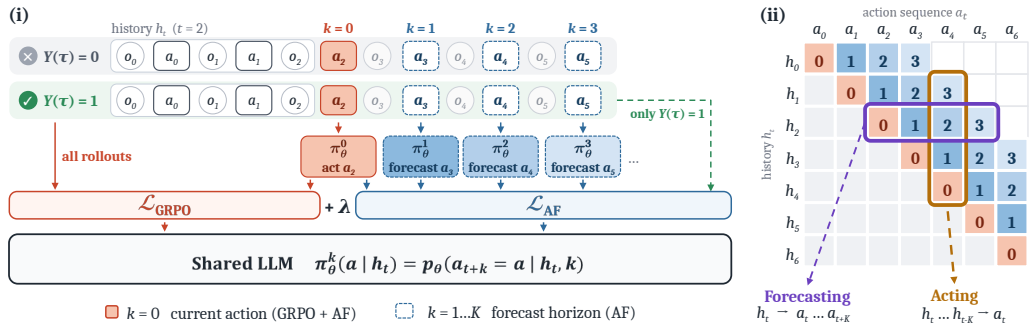

Figure 2: **Overview of CoPE. (i)** Joint training of acting and forecasting. We use one LLM backbone to optimize both, and $\mathcal{L}_{\text{AF}}$ includes $k = 0, \ldots, K$. **(ii)** Co-evolution of acting and forecasting. Acting helps forecasting via trajectory collection and forecasting helps acting via implicit ensembling.

gives the k-step future-action marginal:

$$q_\pi^k(a_{t+k} \mid h_t) = \sum_{\tau_{t:t+k}} \underbrace{\Big[\prod_{j=t}^{t+k-1} \pi(a_j \mid h_j)\, P(o_{j+1} \mid h_j, a_j) \Big]}_{\text{k rounds of closed-loop interaction}} \pi(a_{t+k} \mid h_{t+k}). \tag{3}$$

which equals $\mathbb{E}_{\pi,P}\left[\pi(a \mid H_{t+k}) \mid H_t = h_t\right]$. The product is exactly the loop a world model would have to imagine step by step, with P in place of P_ϕ, reconstructing a full observation at every step. The marginal q_π^k absorbs this loop into a single prediction. Environment dynamics only enter this target through their effects on policy's actions, without reconstructing any intermediate observation.

Parameterizing the marginal. We parameterize q_π^k with the same LLM that acts, defining a horizon-indexed family over a shared backbone:

$$\pi_\theta^k(a_{t+k} \mid h_t) \triangleq p_\theta(a_{t+k} \mid h_t, k) \;\approx\; q_\pi^k(a_{t+k} \mid h_t), \qquad k \in \{0, \ldots, K\}, \tag{4}$$

where k indexes the position within a generated action plan (Sec. 5.2). Since $q_\pi^0 = \pi$ by definition, the horizon-zero member π_θ^0 is the acting policy π_θ of Sec. 3 and is the only distribution executed in the environment. The members with $k \geq 1$ are forecasters of the agent's own future actions, with $\pi = \pi_\theta^0$. We refer to learning $\pi_\theta^{1:K}$ as *multi-step action forecasting*. Compared with Eq. 2, which maps (h_t, a_t) to o_{t+1}, forecasting maps h_t to a_{t+k} for $k = 1, \ldots, K$. The target thus moves from a single step in observation space to up to K steps in action space, addressing the two challenges respectively. Unlike a world model, however, the forecasters are only as meaningful as the policy they describe: for an aimless policy, q_π^k captures only aimless behavior. We therefore condition the forecasting target on task success (Sec. 5.2).

5 CoPE: Co-evolving Acting and Forecasting

CoPE trains a single LLM to both act and forecast its own future actions (Fig. 2). The acting policy and the forecasters share one backbone (Sec. 5.1), the forecasters are trained on successful on-policy trajectories (Sec. 5.2), and the forecasting loss is added to GRPO so that acting and forecasting improve together (Sec. 5.3).

5.1 Sharing Parameters for Acting and Forecasting

The acting policy π_θ^0 and the forecasters $\pi_\theta^{1:K}$ read the same history and output distributions over the same action space, differing only in the horizon k (Eq. 4). We therefore instantiate all of them with a single LLM backbone θ, which brings two benefits. **(1)** No additional model. Forecasting reuses the policy's parameters, so there is no separate model to train, store, or serve. **(2)** Shared representation. Because acting and forecasting update the same parameters, what the forecasting loss learns is written directly into the representation the policy acts from, allowing the two to co-evolve (Sec. 5.3).

Algorithm 1 CoPE training algorithm

Require: Shared backbone θ, horizon K, weight λ

for each iteration **do**
  Collect a group of trajectories $\{\tau_i\}_{i=1}^{G}$ with the acting policy π_θ^0
  $\hat{A}_i \leftarrow (Y(\tau_i) - \mu)/(\sigma + \delta)$ for each i
  $\mathcal{L}_{\mathrm{GRPO}}(\theta) \leftarrow -\frac{1}{G}\sum_{i=1}^{G}\frac{1}{T_i}\sum_t \min\big(\rho_{i,t}\hat{A}_i,\ \mathrm{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon)\hat{A}_i\big)$ ▷ Eq. 1
  $\mathcal{S} \leftarrow \{i : Y(\tau_i) = 1\}$
  $\mathcal{L}_{\mathrm{AF}}(\theta) \leftarrow -\frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}}\sum_t \sum_{\substack{0\le k\le K \\ t+k<T_i}} \log \pi_\theta^k(a_{i,t+k} \mid h_{i,t})$ if $|\mathcal{S}| > 0$ ▷ Eq. 5
  Update θ by minimizing $\mathcal{L}_{\mathrm{GRPO}}(\theta) + \lambda\,\mathcal{L}_{\mathrm{AF}}(\theta)$
end for

5.2 Multi-step Action Forecasting Loss

At each iteration, we collect trajectories with the current acting policy $\pi = \pi_\theta^0$ and hold them fixed during optimization. The marginal q_π^k (Eq. 3) depends on the unknown transition P and cannot be computed, but the realized action a_{t+k} in each rollout is a sample from $q_\pi^k(\cdot \mid h_t)$. We can therefore train all horizon-indexed predictors by maximum likelihood on realized actions, including the current action at $k = 0$ and future actions at $k \geq 1$, without ever modeling P.

Success-filtered multi-step forecasting. As noted in Sec. 4, multi-step forecasting is useful only when the forecast behavior is goal-directed. Forecasting failed trajectories would train the shared LLM backbone to anticipate the actions that GRPO suppresses, sending strong conflicting signals for policy optimization. Therefore, the policy helps collect successful trajectories to construct the forecasting objective, which gives the multi-step action forecasting loss

$$\mathcal{L}_{\mathrm{AF}}(\theta) = -\,\mathbb{E}_{\tau\sim\pi}\Big[Y(\tau)\sum_t\sum_{k=0}^{K}\log \pi_\theta^k(a_{t+k} \mid h_t)\Big], \tag{5}$$

where the sum runs over $k = 0, \ldots, K$ and all t with $t + k < T$. Thus, $\mathcal{L}_{\mathrm{AF}}$ includes both the success-filtered current-action term $\pi_\theta^0(a_t \mid h_t)$ and the future-action terms $\pi_\theta^{1:K}$. Each horizon conditions on h_t alone, without the intervening observations. Evaluating Eq. 5 separately for every horizon would require $K+1$ forward passes over the same history. For efficiency, we instead train all horizons jointly: at history h_t, we append a planning instruction to the prompt and train the model to generate the action sequence $a_{t:t+K}$ in one pass, with the k-th position corresponding to π_θ^k (App. B).

Group normalization of the forecasting loss. While success filtering is essential to CoPE, applying it alone can destabilize training as the policy improves. The number of successful rollouts in RL grows as training progresses, so an unnormalized $\mathcal{L}_{\mathrm{AF}}$ contributes a gradient whose scale grows with the success count and can interfere with the reward-driven term. We therefore average the forecasting loss over the successful rollouts of each group, replacing $Y(\tau)$ in Eq. 5 with $Y(\tau)/|\mathcal{S}_x|$, where $\mathcal{S}_x = \{\tau \in \mathcal{G}_x : Y(\tau) = 1\}$ is the set of successful rollouts in the group $\mathcal{G}_x$ sampled for task x. This removes the direct linear dependence on the number of successes, although the loss magnitude can still vary with trajectory length and token count. Because the reweighting is positive within each task group, it preserves the per-history conditional targets in Eq. 6.

5.3 Co-evolution of Acting and Forecasting

Acting helps forecasting via trajectory collection. For a fixed rollout policy and a sufficiently expressive model, the minimizer of $\mathcal{L}_{\mathrm{AF}}$ satisfies

$$\pi_{\theta^\star}^k(a \mid h_t) = \mathbb{P}(a_{t+k} = a \mid h_t,\, Y = 1) \;\propto\; q_\pi^k(a \mid h_t)\,\mathbb{P}(Y = 1 \mid h_t,\, a_{t+k} = a), \qquad k = 0, \ldots, K. \tag{6}$$

At $k = 0$, this target is the current action in successful trajectories; at $k \geq 1$, it predicts what the agent will do later in those trajectories. Forecasting therefore depends on acting, which supplies the successful rollouts it learns from. As the policy improves, it helps forecasting in two ways. First, it produces more successful rollouts and hence more forecasting targets. Second, the success probability $\mathbb{P}(Y = 1 \mid h_t, a_{t+k} = a)$ approaches one for the actions the policy takes, so the target in Eq. 6 approaches q_π^k, the marginal that Eq. 4 aims to model.

Table 1: **Evaluation results on simulated agent tasks.** For ALFWorld, we report the success rate (%) on each subtask and overall, with a maximum of 30 turns. For WebShop and ScienceWorld, we report the average score and the success rate (%). All RL methods use Qwen2.5-7B-Instruct. Dark gray and light gray mark the best and second-best results in each column across all methods. More details are available in Appendix §C.

Method	ALFWorld							WebShop		ScienceWorld	
	Pick	Look	Clean	Heat	Cool	Pick2	**All**	Score	Succ.	Score	Succ.
Agentic prompting											
Gemini-2.5-flash-lite	82.6	36.8	37.8	20.0	21.4	48.9	46.0	7.8	6.5	13.9	15.0
GPT-5-mini	82.6	47.4	27.0	36.0	14.3	75.6	52.0	11.7	7.0	40.1	43.5
GPT-5.5	97.8	100.0	91.9	88.0	89.3	93.3	93.5	66.1	56.5	66.8	79.0
Qwen2.5-7B	69.6	31.6	29.7	36.0	17.9	11.1	34.0	25.3	2.5	−13.3	2.5
Qwen2.5-7B + Reflexion	84.8	26.3	51.4	44.0	32.1	55.6	54.0	21.9	4.0	−3.7	6.0
RL methods w/o auxiliary loss											
RFT	91.3	73.7	73.0	84.0	64.3	71.1	77.0	51.4	16.0	−11.0	4.5
GRPO	100.0	73.7	94.6	84.0	89.3	88.9	90.5	84.1	64.5	52.8	64.5
GRPO + EMPG	97.8	57.9	89.2	88.0	89.3	95.6	89.5	84.3	67.5	48.2	46.5
RL methods w/ auxiliary loss											
RWML + GRPO	100.0	89.5	89.2	92.0	96.4	84.4	92.0	81.2	64.0	39.4	65.0
ECHO	97.8	89.5	91.9	96.0	96.4	93.3	94.5	85.0	71.5	44.8	67.5
CoPE	100.0	100.0	97.3	96.0	89.3	91.1	95.5	89.4	77.0	53.4	72.5

Forecasting helps acting via implicit ensembling. At the population optimum under success filtering, forecasts of the same action from successive histories are consistent. By the law of total probability,

$$\pi^{k}_{\theta^\star}(\cdot \mid h_t) = \mathbb{E}\big[\, \pi^{k-1}_{\theta^\star}(\cdot \mid h_{t+1}) \;\big|\; h_t,\, Y = 1 \,\big], \qquad k = 1, \ldots, K. \tag{7}$$

Unrolling this recursion, every forecast of an action a_t, whether issued from h_{t-1} or h_{t-K}, targets in expectation the same success-conditioned decision at h_t. The model therefore learns each action from $K{+}1$ points in time, i.e., a column of Fig. 2(ii), rather than from h_t alone as in GRPO. Since all these views share θ, they are implicitly ensembled into the policy.

Joint optimization. We therefore optimize the shared parameters with the weighted sum of the two losses $\mathcal{L}(\theta) = \mathcal{L}_{\mathrm{GRPO}}(\theta) + \lambda\, \mathcal{L}_{\mathrm{AF}}(\theta)$, controlled by a hyper-parameter λ. The details are summarized in Alg. 1.

6 Experimental Settings

Tasks. We evaluate on five long-horizon agent benchmarks with sparse rewards that differ in observation complexity, action space, and horizon. Three are simulated environments: **(1)** ALFWorld (Shridhar et al., 2020) for household tasks, **(2)** ScienceWorld (Wang et al., 2022) for elementary science experiments, and **(3)** WebShop (Yao et al., 2022) for online shopping over long web-page observations. Two are realistic applications: **(1)** τ^2-bench (Barres et al., 2025) for policy-constrained customer service with a user simulator, and **(2)** AppWorld (Trivedi et al., 2024) for composing API calls across consumer apps with code as actions. We use Qwen2.5-7B-Instruct (Qwen et al., 2024) for the three simulated environments and τ^2-bench, and Qwen2.5-14B-Instruct for the more difficult AppWorld tasks. Details are available in Appendix A.

Baselines. On the simulated environments (Table 1), we compare against three groups. (**1**) *Prompting*: Qwen2.5-7B-Instruct, Gemini-2.5-Flash-Lite (Comanici et al., 2025), GPT-5-mini and GPT-5.5 (OpenAI, 2025; 2026), and Reflexion (Shinn et al., 2023), which adds verbal self-reflection on failed trials. (**2**) *RL w/o auxiliary loss*: RFT (Yuan et al., 2023), rejection sampling fine-tuning; GRPO (Shao et al., 2024), critic-free RL with group-normalized advantages; and EMPG (Wang et al., 2025), which rescales GRPO's step-level gradients by policy entropy. (**3**) *RL w/ auxiliary loss*: RWML (Yu et al., 2026), which first trains next-state prediction on environment transitions and then runs GRPO; and ECHO (Shrivastava et al., 2026), which adds a next-observation prediction loss to GRPO. Both supervise only single-step transitions, testing whether forecasting actions over multiple steps is more effective than predicting the next observation. On AppWorld and τ^2-bench (Tables 3 and 2), we include published baseline numbers reported by G2PO (Wang et al., 2026a) and InfoPO (Kong et al.,

2026), together with GRPO and CoPE runs from our implementation. Our direct comparison is between GRPO (ours) and CoPE under our shared setup.

7 Experimental Results

CoPE achieves aggregate gains across five agentic benchmarks. Across both simulated and realistic agentic tasks, CoPE improves the benchmark-level aggregate metrics over GRPO. On ALFWorld (Table 1), CoPE improves performance by 5.0 points over GRPO. The gains are larger on WebShop and ScienceWorld, where success rates increase by 12.5 points to 77.0% and by 8.0 points to 72.5%, respectively; on ScienceWorld, CoPE also achieves the best performance among the RL-based methods. The same trend largely extends to more realistic benchmarks. On τ^2-bench (Table 2), CoPE improves overall success from 27.5% to 28.7%, driven by an approximately 12-point gain on Retail, despite a drop on Airline. We attribute the Airline drop to the abnormal success pattern analyzed in Section 8. Finally, on AppWorld (Table 3), CoPE substantially improves over GRPO on both splits, increasing success from 41.67% to 55.36% on the normal split and from 18.71% to 22.54% on the challenge split.

CoPE outperforms observation-based auxiliary losses on aggregate simulated-task metrics. On the three simulated environments where ECHO and RWML are evaluated, CoPE achieves the best aggregate results. It exceeds the stronger baseline by 1.0 point on ALFWorld, by 4.4 average-score points and 5.5 success-rate points on WebShop, and by 5.0 success-rate points on ScienceWorld. These gains are not uniform across ALFWorld subtasks: ECHO or RWML performs better on Cool and Pick2.

CoPE improves sample efficiency. Figure 1 compares CoPE with GRPO on AppWorld, while Fig. 3 compares CoPE with ECHO and GRPO on ScienceWorld. Overall, CoPE is $1.6\times$ and $2.0\times$ more sample-efficient than GRPO on AppWorld and ScienceWorld, respectively. On ScienceWorld, ECHO requires 53 and 114 steps to reach equivalent performance milestones (making CoPE $1.7\times$ to $1.9\times$ faster). This acceleration is even more pronounced at higher reward tiers: CoPE achieves a 0.6 reward by step 86, a level neither baseline reaches within the 150-step budget. Thus, the multi-step action forecasting loss not only accelerates early-stage learning but also enables the policy to reach a higher performance level under the same budget.

CoPE introduces moderate training overhead. Because CoPE adds a supervised pass over successful rollouts, its direct cost scales with the batch success rate. On ScienceWorld, this pass adds 11% time per training step on average, while shorter rollouts reduce the measured end-to-end step time by 9% compared with GRPO (Fig. 4).

Table 2: **Results on τ^2-bench**. All results are Avg@4 for task success rate with Qwen2.5-7B-Instruct.

Method	Telec.	Retail	Air	Overall
Agentic Prompting				
Qwen2.5	14.4	13.1	7.5	11.7
ReAct	13.1	13.8	10.0	12.3
Published RL results				
RAGEN	17.5	17.5	15.0	16.7
Search-R1	15.6	15.0	10.0	13.5
UserRL	13.8	16.9	7.5	12.7
InfoPO	18.1	18.8	16.3	17.7
Our controlled runs				
GRPO (ours)	38.8	17.5	26.2	27.5
CoPE	40.6	29.4	16.2	28.7

Table 3: **Results on AppWorld**. All results are success rates (%) with Qwen2.5-14B-Instruct.

Method	Normal	Challenge
Agentic Prompting		
Qwen2.5	1.2	0.7
ReAct	10.7	2.6
RL Training		
PPO	19.1	–
RLOO	24.8	–
GiGPO	25.7	–
GRPO	24.8	–
GRPO (ours)	41.7	18.7
CoPE	55.4	22.5

8 Discussions

Q1: What is the empirical evidence for acting and forecasting co-evolution?

In Section 5.3, we analyze the co-evolution mechanism between acting and forecasting. The acting-to-forecasting direction is straightforward: the policy is necessary to collect successful trajectories. Conversely, for the forecasting-to-acting direction, our analysis in Section 5.3 suggests that forecasting improves action selection through implicit ensembling via shared parameters. Here, we provide empirical evidence to support this claim.

Table 4: **Ablations on backbones.** Both model backbones use the same training settings; each cell shows GRPO → CoPE.

Benchmark		Qwen2.5-7B	Qwen3-8B
WebShop	Score	84.1 → **89.4**	83.4 → **87.4**
	Succ.	64.5 → **77.0**	64.0 → **72.5**
ScienceWorld	Score	52.8 → **53.4**	50.1 → **58.3**
	Succ.	64.5 → **72.5**	61.5 → **71.5**

Table 5: **Comparison of action forecasting variants on ScienceWorld.** We report success rate and gain with GRPO.

Method	Succ.	Δ
GRPO + success-filtered SFT	65.5	+1.0
K=0 (no invalid-action filter)	66.5	+2.0
K=0 (invalid-action filter)	69.5	+5.0
CoPE (K=2)	**72.5**	**+8.0**

Table 6: **Explicit ensembling results.** G_{150} and G_{200} denote checkpoints after 150 and 200 training steps.

Setting	G_{150}	G_{200}	CoPE
Top-1 sampling	51.0	61.0	69.0
Top-8 (η=0)	60.0	66.0	**70.0**
Top-8 (η=0.75)	68.5	70.5	69.5
Δ (η=0.75 vs. 0)	+8.5	+4.5	−0.5

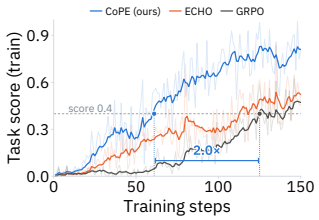

Figure 3: **Analysis of sample efficiency.** CoPE reaches 0.4 in about half the RL steps required by GRPO on ScienceWorld.

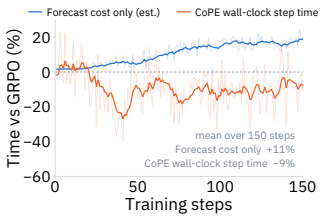

Figure 4: **Analysis on wall-clock efficiency.** Forecasting adds 11% per-step cost, while end-to-end step time decreases by 9%.

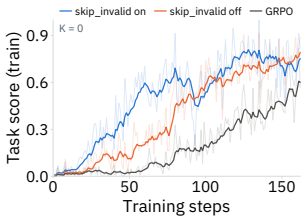

Figure 5: **Invalid-action filtering.** Removing invalid current-action targets improves the sample efficiency of the $K = 0$ baseline.

Evidence for implicit temporal ensembling. For this experiment, we train a separate LoRA forecaster on ScienceWorld trajectories collected by the acting policy while freezing the acting model's parameters. The policy therefore supplies the forecasting data, but forecasting gradients cannot update the policy. We then use the separately parameterized forecaster to rerank candidate actions using forecasts from multiple horizons:

$$\log \pi(a|h_t) = (1-\eta)\, \log \pi_\theta(a \mid h_t) + \frac{\eta}{K} \sum_{i=1}^{K} \log \pi^i_{\theta'}(a|h_{t-i}), \tag{8}$$

where $K = 2$ in this experiment and θ' denotes the LoRA-trained forecaster. We rank eight candidate actions by their combined log-probability and select the highest-ranked action. Within each analysis checkpoint in Table 6, explicit forecasting improves the GRPO policies, whereas applying the same reranking procedure to CoPE provides no gain. This supports, but does not by itself prove, the hypothesis that shared training internalizes part of the temporal ensemble.

Q2: What is the difference between CoPE and GRPO with success-filtered SFT loss?

We first clarify that $\mathcal{L}_{\mathrm{AF}}$ includes the horizon-zero term π^0_θ: augmenting GRPO with a success-filtered SFT loss is therefore approximately equivalent to CoPE with $K = 0$, up to the planning prompt and normalization. We run a prompt-aligned form and report the results in Table 5. At $K = 0$, the auxiliary loss reduces to single-step imitation of successful rollouts. This improves over plain GRPO, but falls significantly short of the full objective, which additionally trains $k = 1, \ldots, K$.

Misalignment of single-step SFT loss. The $K = 0$ objective is prone to overfitting to specific local steps, creating a conflict with the global multi-turn RL objective. In practice, successful trajectories often contain "detours"—suboptimal or invalid actions from which the agent later recovers. Directly imitating these local actions can harm sample efficiency and final convergence. As shown in Fig. 5, removing invalid current-action targets improves the $K = 0$ baseline.

Why multi-step targets help. The full CoPE objective does *not* remove the $k = 0$ term and therefore does not fundamentally bypass detour imitation. Instead, it supplements current-action imitation with future-action targets from $k = 1, \ldots, K$. These additional targets capture how a successful trajectory recovers and progresses after the local step, reducing the relative dominance of any single detour and providing multi-step supervision without task-specific invalid-action filtering. Table 5 shows that this multi-step objective outperforms both the unfiltered and heuristically filtered $K = 0$ variants.

Q3: Why must action forecasting loss be conditioned on success, and what does it learn?

Fundamentally, the action forecasting loss must align with the primary RL objective. Without success conditioning, the model is forced to imitate average or failed trajectories, creating a severe objective

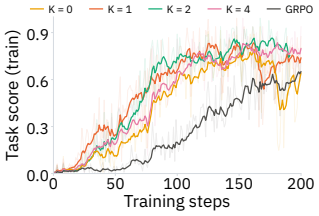

Figure 6: **Ablation on forecast horizon K.** All $K \geq 1$ settings outperform GRPO; $K = 2$ achieves the highest score.

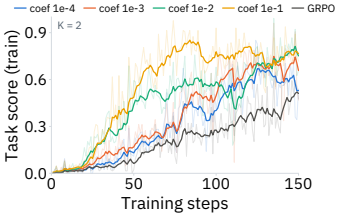

Figure 7: **Ablation on action forecasting weight.** All coefficients outperform the GRPO baseline; $\lambda = 10^{-2}$ achieves the highest.

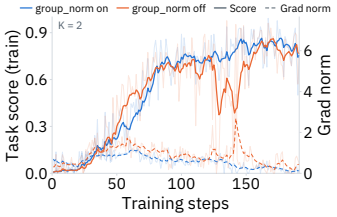

Figure 8: **Ablation on group normalization.** Normalization suppresses spikes in gradient norms and stabilizes late training.

conflict that derails RL training. We hypothesize that the core mechanism of CoPE is to reinforce confirmed multi-horizon patterns—such as specific successful workflows—thereby guiding the model to explore more efficiently during RL optimization.

Abnormal pattern in τ^2-bench. This explains our mixed results on τ^2-bench: a gain on Retail ($17.5\% \rightarrow 29.4\%$), but negative transfer on Airline ($26.2\% \rightarrow 16.2\%$). Success conditioning assumes that successful trajectories contain informative action sequences. Airline weakens this assumption because 38% of its tasks can be solved without taking an action, compared with 4.4% in Retail. Consequently, many successful Airline trajectories contain little useful action supervision, and incidental actions may be reinforced despite not contributing to success.

9 Ablation Studies

Ablation on forecasting horizon K The horizon K is the maximum future offset: the auxiliary plan contains the current-action target at $k = 0$ and future-action targets through $k = K$. Short horizons keep the future targets close to the current history but capture less multi-step structure. Conversely, long horizons introduce irreducible noise, as the k-step marginal q_π^k becomes less correlated with the current history h_t. In Fig. 6, we compare the $K = 0$ success-filtered SFT baseline with forecasting horizons $K \in \{1, 2, 4\}$. Every $K \geq 1$ setting outperforms GRPO and the $K = 0$ baseline. $K = 2$ performs best, while $K = 4$ declines slightly as noise from distant predictions begins to dominate.

Ablation on forecasting coefficient λ Another important hyperparameter is the coefficient λ for the action forecasting loss. In Fig. 7, we sweep λ from 10^{-4} to 10^{-1} while fixing $K = 2$. Notably, every tested value of λ consistently improves upon the GRPO baseline throughout training. We find that λ primarily controls the pace of learning: larger values accelerate early gains, whereas smaller values follow a similar but delayed trajectory. The two largest coefficients converge to similar peak scores, while the smaller ones still remain above GRPO. We adopt $\lambda = 10^{-2}$ for all agentic tasks.

Ablation on group normalization. As discussed in Sec. 5, the number of successful rollouts within a group naturally increases as the policy improves. Without normalization, the accumulated forecasting gradient scales directly with this success count. This shifting balance between the primary RL objective and the forecasting auxiliary loss causes spikes in gradient norms and reward fluctuations. Averaging over successful rollouts removes this direct dependence, although the resulting loss can still vary with trajectory length. As shown in Fig. 8, normalization reduces these spikes.

Ablation on model backbone The results in Table 1 use Qwen2.5-7B-Instruct, leaving open whether the gain is specific to that backbone. We repeat the comparison against GRPO on WebShop and ScienceWorld with Qwen3-8B (Yang et al., 2025) and report it alongside the Qwen2.5-7B results. In Table 4, CoPE improves both score and success rate on both benchmarks under both backbones, indicating that its gains are not tied to a particular model generation.

10 Conclusion

We introduced CoPE, which trains a language agent to forecast its own future actions alongside acting. Rather than fitting noisy observations, CoPE trains on the success-filtered action plan $a_{t:t+K}$, combining the current-action target with forecasts up to K steps in the future. Across five benchmarks, CoPE improves aggregate results over RL baselines. It also improves sample efficiency with moderate computational overhead. We hope this work motivates further methods for sample-efficient long-horizon agentic RL.

Reproducibility Statement

We describe the CoPE algorithm in Section 5 and Algorithm 1, and the tasks, models, baselines, and evaluation protocols in Section 6. Appendix §A records model licenses and software dependencies, while Appendix §C lists the configurations currently documented in the manuscript. Evaluation metrics are defined in Appendix §C.3. We will release the complete per-method configuration files and training code upon publication.

Ethics Statement

This work introduces CoPE, a training algorithm for improving the sample efficiency of reinforcement learning for LLM-based agents; it is a methodological contribution rather than a deployed system, and its societal impact is largely mediated by how downstream practitioners apply it. All experiments are conducted on standard agentic benchmarks (ALFWorld, ScienceWorld, WebShop, τ^2-bench, and AppWorld) using open-weight or proprietary models strictly for research evaluation; the work does not involve human subjects, crowdsourcing, or the collection of new personal data. By coupling action forecasting and policy optimization within a single backbone, CoPE can reduce environment-interaction costs without training or serving a separate world model, lowering the barrier for resource-constrained groups to study long-horizon LLM agents. We also note that, like other advances in sample-efficient agentic RL, more capable autonomous agents could in principle be misused for tasks such as automated misinformation or unauthorized web automation; practitioners building on CoPE in higher-stakes domains should pair it with appropriate safety evaluation, deployment-time monitoring, and human oversight.

AI Use Statement

In this work, we used generative AI tools to assist with method development and proof writing; to provide feedback on experimental design, including the separated-forecaster and implicit-ensembling analysis in Section 8; to help interpret experimental results; and to implement parts of the training and evaluation code. We did not use generative AI tools to generate synthetic datasets, clean or reformat datasets, perform translation, or conduct qualitative data analysis. We also used generative AI tools to draft and edit text for clarity, suggest section organization, search for and summarize related literature, and format references. The authors reviewed the AI-assisted derivations, code, reported results, and citations and take responsibility for the final content.

References

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

Markus J. Buehler. Generative retrieval-augmented ontologic graph and multiagent strategies for interpretive large language model-based materials design. *ACS Engineering Au*, 4:241–277, 2024. URL https://pubs.acs.org/aeacb3/article/4/2/241/351595/Generative-Retrieval-Augmented-Ontologic-Graph-and.

Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. *arXiv preprint arXiv:2410.13232*, 2024. URL https://arxiv.org/abs/2410.13232.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/7f489f642a0ddb10272b5c31057f0663-Abstract.html.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Sara Cao, Yuanhao Xiong, Haibo Tong, Huaxiu Yao, Hengduo Li, Jiacheng Zhu, Xian Li, Dawn Song, Bo Li,

Jason Weston, and Dat Huynh. Scaling agent learning via experience synthesis. *arXiv preprint arXiv:2511.03773*, 2025.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin C. M. Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.026. URL https://www.roboticsproceedings.org/rss19/p026.html.

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

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*, 2023. URL https://arxiv.org/abs/2306.06070.

Hang Ding, Peidong Liu, Junqiao Wang, Ziwei Ji, Meng Cao, Rongzhao Zhang, Lynn Ai, Eric Yang, Tianyu Shi, and Lei Yu. DynaWeb: Model-based reinforcement learning of web agents. *arXiv preprint arXiv:2601.22149*, 2026.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. *arXiv preprint arXiv:2505.10978*, 2025. URL https://arxiv.org/abs/2505.10978.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15706–15734. PMLR, 2024. URL https://proceedings.mlr.press/v235/gloeckle24a.html.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. MetaGPT: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352v1*, 2023. URL https://arxiv.org/abs/2308.00352v1.

Michael Y. Hu, Benjamin Van Durme, Jacob Andreas, and Harsh Jhamtani. Sample-efficient online learning in LM agents via hindsight trajectory rewriting. *arXiv preprint arXiv:2510.10304*, 2025. URL https://arxiv.org/abs/2510.10304.

Fanqi Kong, Jiayi Zhang, Mingyi Deng, Chenglin Wu, Yuyu Luo, and Bang Liu. InfoPO: Information-driven policy optimization for user-centric agents. *arXiv preprint arXiv:2603.00656*, 2026.

Jessy Lin, Yuqing Du, Olivia Watkins, Danijar Hafner, Pieter Abbeel, Dan Klein, and Anca Dragan. Learning to model the world with language. *arXiv preprint arXiv:2308.01399*, 2023. URL https://arxiv.org/abs/2308.01399.

Tennison Liu and Mihaela van der Schaar. Truly self-improving agents require intrinsic metacognitive learning. *arXiv preprint arXiv:2506.05109*, 2025. URL https://arxiv.org/abs/2506.05109.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024. URL https://arxiv.org/abs/2408.06292.

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.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. Agent lightning: Train ANY AI agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025. URL https://arxiv.org/abs/2508.03680.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3878–3887. PMLR, 2018. URL https://proceedings.mlr.press/v80/oh18b.html.

OpenAI. GPT-5 system card, 2025. URL https://openai.com/index/gpt-5-system-card/.

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

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115v1*, 2024. URL https://arxiv.org/abs/2412.15115v1.

Ranjan Sapkota, Konstantinos I. Roumeliotis, and Manoj Karkee. Vibe coding vs. agentic coding: Fundamentals and practical implications of agentic ai. *arXiv preprint arXiv:2505.19443*, 2025. URL https://arxiv.org/abs/2505.19443.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020. URL https://arxiv.org/abs/2010.03768.

Vaishnavi Shrivastava, Piero Kauffmann, Ahmed Awadallah, and Dimitris Papailiopoulos. ECHO: Terminal agents learn world models for free. *arXiv preprint arXiv:2605.24517*, 2026.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024. URL https://aclanthology.org/2024.acl-long.850/.

Jiawei Wang, Jiacai Liu, Yuqian Fu, Yingru Li, Xintao Wang, Yuan Lin, Yu Yue, Lin Zhang, Yang Wang, and Ke Wang. Harnessing uncertainty: Entropy-modulated policy gradients for long-horizon llm agents. *arXiv preprint arXiv:2509.09265*, 2025. URL https://arxiv.org/abs/2509.09265.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022. URL https://aclanthology.org/2022.emnlp-main.775/.

Yunan Wang, Minghui Song, Zihan Zhang, Shaohan Huang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. Group-graph policy optimization for long-horizon agentic reinforcement learning. *arXiv preprint arXiv:2606.22995*, 2026a.

Zixuan Wang, Yuchen Yan, Hongxing Li, Teng Pan, Dingming Li, Ruiqing Zhang, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Milestone-guided policy learning for long-horizon language agents. *arXiv preprint arXiv:2605.06078*, 2026b. URL https://arxiv.org/abs/2605.06078.

Jinyang Wu, Shuo Yang, Zhengxi Lu, Fan Zhang, Yuhao Shen, Lang Feng, Haoran Luo, Zheng Lian, Shuai Zhang, Zhengqi Wen, and Jianhua Tao. SEED: Self-evolving on-policy distillation for agentic reinforcement learning. *arXiv preprint arXiv:2607.14777*, 2026. URL https://arxiv.org/abs/2607.14777.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025. URL https://arxiv.org/abs/2504.08066.

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

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

Shuo Yang, Jinyang Wu, Zhengxi Lu, Yuhao Shen, Fan Zhang, Lang Feng, Shuai Zhang, Haoran Luo, Zheng Lian, Zhengqi Wen, and Jianhua Tao. OPID: On-policy skill distillation for agentic reinforcement learning. *arXiv preprint arXiv:2606.26790*, 2026. URL https://arxiv.org/abs/2606.26790.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *arXiv preprint arXiv:2207.01206*, 2022. URL https://arxiv.org/abs/2207.01206.

Xiao Yu, Baolin Peng, Ruize Xu, Yelong Shen, Pengcheng He, Suman Nath, Nikhil Singh, Jianfeng Gao, and Zhou Yu. Reinforcement world model learning for LLM-based agents. *arXiv preprint arXiv:2602.05842*, 2026.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, Jian Xie, Yuxuan Sun, Boyu Gou, Qi Qi, Zihang Meng, Jianwei Yang, Ning Zhang, Xian Li, Ashish Shah, Dat Huynh, Hengduo Li, Zi Yang, Sara Cao, Lawrence Jang, Shuyan Zhou, Jiacheng Zhu, Huan Sun, Jason Weston, Yu Su, and Yifan Wu. Agent learning via early experience. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2510.08558.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. WebPilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(22):23378–23386, 2025. doi: 10.1609/aaai.v39i22.34505. URL https://ojs.aaai.org/index.php/AAAI/article/view/34505.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.016. URL https://www.roboticsproceedings.org/rss19/p016.html.

Binbin Zheng, Zijun Xie, Guanqun Zhao, Enlei Gong, Xing Ma, Xiaoliang Fu, and Zeyu Chen. Group-reflective self-distillation for agentic reinforcement learning. *arXiv preprint arXiv:2607.28076*, 2026. URL https://arxiv.org/abs/2607.28076.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024. URL https://arxiv.org/abs/2401.01614.

Tianshi Zheng, Zheye Deng, Hong Ting Tsang, Weiqi Wang, Jiaxin Bai, Zihao Wang, and Yangqiu Song. From automation to autonomy: A survey on large language models in scientific discovery. In *Conference on Empirical Methods in Natural Language Processing*, 2025. URL https://aclanthology.org/2025.emnlp-main.895/.

A Artifact Details

A.1 Model License

GPT-5-mini License: Proprietary (OpenAI)
GPT5-5.5 License: Proprietary (OpenAI)
Gemini-2.5-flash-lite Proprietary (Google)
Qwen2.5-7B-Instrct License: Apache 2.0
Qwen2.5-3B-Instrct License: Apache 2.0

A.2 Software Versions

For our simulated agentic environments, we build upon AgentGym-RL[1], which provides unified task configurations for ALFWorld, WebShop, and SciWorld. Our training infrastructure is built on top of verl[2] (v0.2.0.post2).

A.3 Dataset Statistics

Table 7 gives the size of every split we use.

Table 7: **Splits used in this work.** "Pool" is the number of distinct tasks the environment can instantiate at all, from which the two splits are drawn; the splits do not have to exhaust it. Counts are measured, not quoted.

Environment	Train	Test	Pool	Unit of a task
ALFWorld	2,420	200	2,620	TextWorld game file
WebShop	3,930	200	6,910	shopping instruction (goal)
SciWorld	2,120	200	4,639	(task type, variation) pair
τ^2	178	100	278	customer-service task
AppWorld	90	168/417	732	task in a scenario

Table 8: **ALFWorld subtask families.** Derived from the task type encoded in each game file's name, so the two splits are counted the same way.

Split	All	Pick	Look	Clean	Heat	Cool	Pick2
Train	2,420	672	196	448	321	394	389
Test	200	46	19	37	25	28	45

ALFWorld. The training pool is the 2,420 games under `json_2.1.1/train` and the test set is the 200 games under `json_2.1.1/valid_train`; the environment server concatenates the two mapping files into one list, so test ids are simply indices 2420–2619 into it. The splits are therefore drawn from different game-file directories and share no game. Tasks fall into six families, read off the task type in the file name (Table 8): *Pick* (place an object), *Look* (examine an object under a lamp), *Clean*, *Heat*, *Cool*, and *Pick2* (place two objects). The test split keeps roughly the training proportions, the largest gap being *Pick2* (22.5% of test against 16.1% of train).

WebShop. Our WebShop instance is configured with 1,000 products, not the full catalogue: the environment server is constructed with `num_products=1000` and the product files are the 1K subsets, which we verified are byte-equivalent to the first 1,000 entries of the full files. Goals are the *synthetic* ones generated from those products' attributes (the server is not run in human-goal mode), which yields 6,910 instructions; the generator is seeded (`random.seed(233)` before the shuffle), so an id denotes the same instruction in every run and on every machine. We train on 3,930

[1] `https://github.com/WooooDyy/AgentGym-RL`
[2] `https://github.com/volcengine/verl`

of them and test on 200. The two sets are disjoint and together cover 59.8% of the pool. Note that the commonly quoted WebShop figures — 12,087 human-written instructions, a 500-instruction test set, the full product catalogue — describe a different configuration and are not comparable to ours.

SciWorld. SciWorld is our abbreviation for ScienceWorld. The environment defines 30 task types; the server excludes seven of them (task ids `5-1`, `5-2`, `9-1`, `9-2`, `9-3`, `10-1`, `10-2`) and enumerates every variation of the remaining 23, giving a pool of 4,639 (task type, variation) pairs indexed by a single running id. We train on 2,120 of these and test on 200. The training ids cover 22 of the 23 task types and the test ids cover 12.

τ^2-Bench. τ^2-Bench ships its own train/test partition per domain, and we use all three domains together: retail (74 train, 40 test), airline (30, 20) and telecom (74, 40), i.e. 178 training and 100 test tasks. We first compute the metric separately for each domain and report their unweighted arithmetic mean, so retail, airline, and telecom contribute equally despite their different test-set sizes. The environment server is launched with `TAU2_DOMAIN=retail+airline+telecom` and the split selected by `TAU2_TASK_SPLIT`; task ids are positional across the concatenated domains in that order, and the id files are generated from the same loaders the server uses (`scripts/make_tau2_itemid.py`), so a row cannot name one task and train on another. Each task pairs a customer goal with a database state and a set of state-based assertions, and every episode also runs a simulated customer, so a τ^2 task is substantially more expensive than an ALFWorld or WebShop one — which is why the pool is three orders of magnitude smaller.

AppWorld. AppWorld ships four splits and we use its own partition unchanged, read with `appworld.load_task_ids`: `train` (90 tasks), `dev` (57), `test_normal` (168) and `test_challenge` (417), 732 in total. We train on `train` and evaluate on both test splits, which the evaluation harness runs as separate passes because the environment servers have to be restarted per split (`APPWORLD_SPLIT` selects the task list). `test_challenge` is not a harder sample of the same distribution: it is built from scenarios held out entirely, so it measures transfer to unseen app combinations rather than to unseen instances of seen ones. An AppWorld task is a Python REPL session against nine simulated apps and is scored by state-based unit tests, so success is all-or-nothing per task; with 90 training tasks it is by far the smallest training pool of the five environments.

B Consistency of the Plan Implementation

Sec. 5 defines the forecasters π_θ^k per horizon (Eq. 4) and trains them with $\mathcal{L}_{\mathrm{AF}}$ (Eq. 5). In practice, all horizons are trained jointly as a single plan: at history h_t, a planning instruction is appended to the prompt and the model generates the actions $a_{t:t+K}$ autoregressively,

$$\mu_\theta(a_{t:t+K} \mid h_t) = \prod_{k=0}^{K} \mu_\theta(a_{t+k} \mid h_t,\ a_{t:t+k-1}), \tag{9}$$

trained with

$$\mathcal{L}_{\mathrm{plan}}(\theta) = -\,\mathbb{E}_{\tau\sim\pi}\Big[Y(\tau)\sum_t \log \mu_\theta(a_{t:t+K} \mid h_t)\Big]. \tag{10}$$

We identify the forecaster $\pi_\theta^k(\cdot \mid h_t)$ with the marginal of the k-th plan position, $\pi_\theta^k(a \mid h_t) = \sum_{a_{t:t+K}:\, a_{t+k}=a} \mu_\theta(a_{t:t+K} \mid h_t)$, and show that the two objectives share the same population optimum.

Claim. For a fixed rollout policy and a sufficiently expressive model, at every history h_t with $p(h_t, Y=1) > 0$, the minimizer of Eq. 10 satisfies $\mu_{\theta^\star}(a_{t:t+K} \mid h_t) = p(a_{t:t+K} \mid h_t, Y=1)$, and therefore

$$\pi_{\theta^\star}^k(a \mid h_t) = p(a_{t+k} = a \mid h_t, Y=1), \qquad k = 0, \dots, K, \tag{11}$$

which is the minimizer of $\mathcal{L}_{\mathrm{AF}}$ in Eq. 5.

Proof. Since $Y(\tau) \in \{0,1\}$, Eq. 10 equals $p(Y=1)$ times the expected loss over successful trajectories. Grouping its terms by history, each history h_t enters with a positive weight (its expected number of visits in successful trajectories) and contributes the cross-entropy $\mathbb{E}_{a_{t:t+K}\sim p(\cdot|h_t,\,Y=1)}\big[-\log \mu_\theta(a_{t:t+K} \mid h_t)\big]$. By Gibbs' inequality, each term is minimized uniquely at $\mu_\theta(\cdot \mid h_t) = p(\cdot \mid$

$h_t, Y = 1)$, which an autoregressive model can represent since Eq. 9 is the chain rule. Distinct histories determine separate conditionals, so the minimizer holds at every such h_t. Marginalizing both sides over all positions except k gives Eq. 11. Eq. 5, in turn, is a sum of per-horizon cross-entropies whose targets are exactly these marginals.

C Experimental Details

C.1 Baseline Methods

We compare our method with several baseline training paradigms, including GRPO, RWML, ECHO, and EMPG. These methods represent different ways of improving agent policies, ranging from online reinforcement learning to world-model-based supervision and offline reward-guided optimization.

GRPO. GRPO is an online reinforcement learning baseline. For each training query, the actor samples a group of trajectories from the current policy and optimizes the policy using group-relative advantages. Different from critic-based PPO, GRPO estimates advantages within each sampled group and therefore avoids training an additional value model. In our experiments, GRPO is applied to ALFWorld, WebShop, SciWorld, τ^2-Bench, and AppWorld.

RWML. RWML is a world-model-based learning baseline that uses offline trajectory triplets and reward signals derived from world-model matching. Unlike GRPO, which collects online rollouts from the environment, RWML relies on pre-collected transition data and uses the learned world-model signal to guide policy optimization. We pretrain its world model before applying GRPO policy optimization and evaluate it on ALFWorld, WebShop, and SciWorld.

ECHO. ECHO (Environment Cross-entropy Hybrid Objective) (Shrivastava et al., 2026) is a world-model-augmented reinforcement learning baseline that combines the GRPO loss on action tokens with an auxiliary cross-entropy loss on environment-observation tokens. It reuses the same rollouts, model backbone, and forward-pass logits as policy optimization, thereby providing dense predictive supervision without additional environment interactions or a separately pretrained world model. Unlike RWML, which learns its world model from offline trajectory data before policy optimization, ECHO couples observation-space world modeling directly with online policy updates. We use the same GRPO hyperparameters for ECHO and set the environment-prediction coefficient to 0.01 on ALFWorld, WebShop, and SciWorld.

EMPG. EMPG is an entropy-modulated policy gradient baseline that re-calibrates the learning signal based on step-wise predictive uncertainty. Unlike standard policy gradients, where update magnitude is implicitly coupled with action entropy, EMPG amplifies updates for confident correct actions, penalizes confident errors, and attenuates updates from uncertain steps. An additional future-clarity bonus encourages the agent toward lower-entropy next states. In our experiments, we apply EMPG on top of GRPO and evaluate it on ALFWorld, WebShop, and SciWorld.

C.2 Training Hyperparameters

We summarize the main training hyperparameters of all methods in Table 9. Unless otherwise specified, the settings follow the default configurations in our training scripts.

C.3 Evaluation Metrics

Here we define the evaluation metrics used in our experiments. For each benchmark, we evaluate the agent on a set of N task instances. For the i-th instance, the environment returns an environment-specific scalar score r_i and a binary success indicator $s_i \in \{0, 1\}$. We report the average score and success rate:

$$\text{Score} = \frac{1}{N}\sum_{i=1}^{N} r_i, \qquad \text{Succ} = \frac{1}{N}\sum_{i=1}^{N} s_i. \tag{12}$$

	GRPO			ECHO			RWML			EMPG		
Hyperparameter	**ALFWorld**	**WebShop**	**SciWorld**	**ALFWorld**	**WebShop**	**SciWorld**	**ALFWorld**	**WebShop**	**SciWorld**	**ALFWorld**	**WebShop**	**SciWorld**
Learning rate	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}
KL coefficient	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}
ECHO coefficient λ	–	–	–	0.01	0.01	0.01	–	–	–	–	–	–
Training batch size	8	16	16	8	16	16	16	16	16	8	16	16
PPO mini-batch size	2	8	8	2	8	8	8	8	8	2	8	8
Micro-batch size per GPU	1	1	1	1	1	1	1	1	1	1	1	1
Rollout group size n	8	8	8	8	8	8	8	8	8	8	8	8
PPO epochs	2	2	1	2	2	1	2	2	2	2	2	2
Total epochs	2	2	10	2	2	10	2	2	2	2	2	2
Max rounds	30	15	20	30	15	20	30	15	20	30	15	20
Max prompt length	1024	768	1024	1024	768	1024	1024	768	1024	1024	768	1024
Max response length	6000	8192	4096	6000	8192	4096	6000	8192	4096	6000	8192	4096
Max model length	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384
Rollout temperature	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table 9: **Details of training hyperparameters**. We include experiments for GRPO, ECHO, RWML, and EMPG. The table is organized first by method and then by dataset.

Score. The score measures the average environment reward or normalized task score. Different from purely binary task completion, some benchmarks provide graded feedback that reflects partial progress during task execution. For example, an agent may receive non-zero reward for completing intermediate steps even if the final task is not fully solved. Therefore, we report the average score to capture such process-level progress.

Success rate. The success rate measures the fraction of tasks that are fully completed. A task is counted as successful only when the environment evaluator determines that the final goal has been achieved. This metric directly reflects end-to-end task completion ability.

For benchmarks with subcategories, we compute the same metrics within each group. Except for τ^2-Bench, the overall result is computed over the full evaluation set and is therefore a count-weighted average over all instances. For τ^2-Bench, we instead report the unweighted arithmetic mean of the retail, airline, and telecom domain-level results.

D Environment Prompts

Every episode is one chat sequence. It opens with the environment's standing instruction as the first *user* turn and a fixed assistant acknowledgement; the environment's first observation then arrives as the next user turn, and from there user (observation) and assistant (action) turns alternate. The blocks below give each environment's opening verbatim, and mark with `{history}` the point at which every completed $(\text{action}_t,\ \text{observation}_{t+1})$ pair is inserted. The sequence is extended in place rather than re-rendered each turn, so `{history}` is literally the tokens already in the context, not a re-serialisation of them. Different from our notations in the main content, K refers to the number of actions in our plan. It operates simply as a offset and does not introduce any inconsistency between our implementation and method.

Conversation layout, shared by all four environments

```
[USER]
<the environment's standing instruction -- verbatim in the blocks below>

[ASSISTANT]
<the fixed acknowledgement -- verbatim in the blocks below>

[USER]
<first observation from the environment>

{history}

[USER]
{observation}

[ASSISTANT]
<- the policy generates the action here
```

Rendered with the chat template the tokeniser actually emits, one round is

One round, as tokenised

```
<|im_start|>assistant\n{action}<|im_end|>\n<|im_start|>user\n{observation}<|im_end|>\n
```

and `{history}` above stands for the concatenation of those rounds.

D.1 ALFWORLD

ALFWorld — standing instruction (first user turn)

```
Interact with a household to solve a task. Imagine you are an intelligent agent in a
↪ household environment and your target is to perform actions to complete the task goal.
↪ At the beginning of your interactions, you will be given the detailed description of the
↪  current environment and your goal to accomplish. For each of your turn, you will be
↪ given a list of actions which you can choose one to perform in this turn. You should
↪ choose from two actions: "THOUGHT" or "ACTION". If you choose "THOUGHT", you should
↪ first think about the current condition and plan for your future actions, and then
↪ output your action in this turn. Your output must strictly follow this format:"Thought:
your thoughts.

Action:
your next action"; If you choose "ACTION", you should directly output the action in this
↪ turn. Your output must strictly follow this format:"Action:
your next action". After your each turn, the environment will give you immediate feedback
↪ based on which you plan your next few steps. if the envrionment output "Nothing happened
↪ ", that means the previous action is invalid and you should try more options.
 Reminder:
1. the action must be chosen from the given available actions. Any actions except provided
↪ available actions will be regarded as illegal.
2. Think when necessary, try to act directly more in the process.
```

ALFWorld — fixed assistant acknowledgement

```
OK. I'll follow your instructions and try my best to solve the task.
```

D.2 WEBSHOP

WebShop — standing instruction (first user turn)

```
You are web shopping.
I will give you instructions about what to do.
You have to follow the instructions.
Every round I will give you an observation and a list of available actions, you have to
↪ respond an action based on the state and instruction.
You can use search action if search is available.
You can click one of the buttons in clickables.
An action should be of the following structure:
search[keywords]
click[value]
If the action is not valid, perform nothing.
Keywords in search are up to you, but the value in click must be a value in the list of
↪ available actions.
Remember that your keywords in search should be carefully designed.
Your response should use the following format:

Thought:
I think ...

Action:
click[something]
```

WebShop — fixed assistant acknowledgement

```
Ok.
```

D.3 SCIWORLD

The per-episode goal ("Your task is to measure the melting point of tin …") is not part of the instruction: it arrives inside the first observation.

SciWorld — standing instruction (first user turn)

```
You are an agent for science world. Every round I will give you an observation, you have to
↪  respond an action based on the observation to finish the given task. Here are the
↪ actions you may take: [{"action": "open/close OBJ", "description": "open/close a
↪ container"}, {"action": "de/activate OBJ", "description": "activate/deactivate a device
↪ "}, {"action": "connect OBJ to OBJ", "description": "connect electrical components"}, {"
↪ action": "disconnect OBJ", "description": "disconnect electrical components"}, {"action
↪ ": "use OBJ [on OBJ]", "description": "use a device/item"}, {"action": "look around", "
↪ description": "describe the current room"}, {"action": "look at OBJ", "description": "
↪ describe an object in detail"}, {"action": "look in OBJ", "description": "describe a
↪ container's contents"}, {"action": "read OBJ", "description": "read a note or book"}, {"
↪ action": "move OBJ to OBJ", "description": "move an object to a container"}, {"action":
↪ "pick up OBJ", "description": "move an object to the inventory"}, {"action": "put down
↪ OBJ", "description": "drop an inventory item"}, {"action": "pour OBJ into OBJ", "
↪ description": "pour a liquid into a container"}, {"action": "dunk OBJ into OBJ", "
↪ description": "dunk a container into a liquid"}, {"action": "mix OBJ", "description": "
↪ chemically mix a container"}, {"action": "go to LOC", "description": "move to a new
↪ location"}, {"action": "eat OBJ", "description": "eat a food"}, {"action": "flush OBJ",
↪ "description": "flush a toilet"}, {"action": "focus on OBJ", "description": "signal
↪ intent on a task object"}, {"action": "wait", "description": "take no action for 10
↪ iterations"}, {"action": "wait1", "description": "take no action for 1 iteration"}, {"
↪ action":"examine OBJ","description":"provides a description of the objects present on or
↪  in a receptacle."}, {"action": "task", "description": "describe current task"}, {"
↪ action": "inventory", "description": "list your inventory"}]
Your response should use the following format:
Thought:
your thoughts.

Action:
your next action
```

SciWorld — fixed assistant acknowledgement

```
OK. I'll follow your instructions and try my best to solve the task.
```

D.4 APPWORLD

AppWorld's instruction is by far the longest of the four (12,396 characters) because it embeds a complete worked example. The `ASSISTANT:` and `USER:` lines inside it are part of that *static* demonstration, not dialogue history — the real history still goes where `{history}` is marked in the layout above. The task itself is delivered in the first observation, as the last line of the instruction announces.

AppWorld — standing instruction (first user turn)

````
I am your supervisor, and you are an AI Assistant whose job is to complete my day-to-day
↪ tasks fully autonomously.

To do this, you will need to interact with app(s) (e.g., spotify, venmo etc) using their
↪ associated APIs on my behalf. For this you will undertake a *multi-step conversation*
↪ using a python REPL environment. That is, you will write the python code, the
↪ environment will execute it and show you the result, based on which, you will write
↪ python code for the next step and so on, until you've achieved the goal. This
↪ environment will let you interact with app(s) using their associated APIs on my behalf.

Here are three key APIs that you need to know to get more information

# To get a list of apps that are available to you.

```python
print(apis.api_docs.show_app_descriptions())
```

# To get the list of APIs under any app listed above, e.g. spotify
````

````
```python
print(apis.api_docs.show_api_descriptions(app_name='spotify'))
```

# To get the specification of a particular api, e.g. spotify app's login api

```python
print(apis.api_docs.show_api_doc(app_name='spotify', api_name='login'))
```

Each code execution will produce an output that you can use in subsequent calls. Using these APIs, you can now generate code, that I will execute, to solve the task. Let's start with the task

Task: How many playlists do I have in my Spotify playlist library?

ASSISTANT:
Okay. Let's first find which APIs are available to use in Spotify.

```python
print(apis.api_docs.show_api_descriptions(app_name='spotify'))
```

USER:
Output:
```
[
 ...
 "login : Login to your account.",
 "logout : Logout from your account.",
 ...
]
```

ASSISTANT:
Okay. Looks like I can use the `login` api. Let's find its specifications.

```python
print(apis.api_docs.show_api_doc(app_name='spotify', api_name='login'))
```

USER:
Output:
```
{
 "app_name": "spotify",
 "api_name": "login",
 "path": "/spotify/auth/token",
 "method": "POST",
 "description": "Login to your account.",
 "parameters": [
  {
   "name": "username",
   "type": "string",
   "required": true,
   "description": "Your account email.",
   "default": null,
   "constraints": []
  },
  {
   "name": "password",
   "type": "string",
   "required": true,
   "description": "Your account password.",
   "default": null,
   "constraints": []
  }
 ],
 "response_schemas": {
  "success": {
   "access_token": "string",
   "token_type": "string"
  },
  "failure": {
   "message": "string"
  }
 }
}
````
````

````
```

ASSISTANT:
I need the supervisor's username and password. Let's see if any app can help me get that.

```python
print(apis.api_docs.show_app_descriptions())
```

USER:
Output:
```
[
 {
  "name": "api_docs",
  "description": "An app to search and explore API documentation."
 },
 {
  "name": "supervisor",
  "description": "An app to access supervisor's personal information, account credentials,
↪ addresses, payment cards, and manage the assigned task."
 },
 {
  "name": "amazon",
  "description": "An online shopping app to buy products and manage orders, returns, etc."
 },
 {
  "name": "phone",
  "description": "An app to find and manage contact information for friends, family members
↪ , etc., send and receive messages, and manage alarms."
 },
 {
  "name": "file_system",
  "description": "A file system app to create and manage files and folders."
 },
 {
  "name": "spotify",
  "description": "A music streaming app to stream songs and manage song, album and playlist
↪  libraries."
 },
 {
  "name": "venmo",
  "description": "A social payment app to send, receive and request money to and from
↪ others."
 },
 {
  "name": "gmail",
  "description": "An email app to draft, send, receive, and manage emails."
 },
 {
  "name": "splitwise",
  "description": "A bill splitting app to track and split expenses with people."
 },
 {
  "name": "simple_note",
  "description": "A note-taking app to create and manage notes"
 },
 {
  "name": "todoist",
  "description": "A task management app to manage todo lists and collaborate on them with
↪ others."
 }
]
```

ASSISTANT:
Looks like the supervisor app could help me with that.

```python
print(apis.api_docs.show_api_descriptions(app_name='supervisor'))
```

USER:
Output:
```
[
 ...
 "show_account_passwords : Show your supervisor's account passwords."
 ...
````

```
]
```

ASSISTANT:
I can use `show_account_passwords` to get the passwords. So I will call that and save it in a variable for future reference.

```python
passwords=apis.supervisor.show_account_passwords()
print(passwords)
```

USER:
Output:
```
[
 {
  "account_name": "spotify",
  "password": "dummy_spotify_pass"
 },
 {
  "account_name": "file_system",
  "password": "dummy_fs_pass"
 },
 ...
]
```

ASSISTANT:
So the Spotify password is an entry in the `passwords` list with the account_name=spotify. The login username is the supervisor's email, which I must look up via the supervisor app rather than assume.

```python
spotify_password = [p for p in passwords if p["account_name"] == "spotify"][0]["password"]
supervisor_email = apis.supervisor.show_profile()["email"]
login_result = apis.spotify.login(username=supervisor_email, password=spotify_password)
print(login_result)
```

USER:
Output:
```
{
 "access_token": "fake_access_token",
 "token_type": "Bearer"
}
```

ASSISTANT:
First, let's check the available APIs for the Spotify app.

```python
print(apis.api_docs.show_api_descriptions(app_name='spotify'))
```

USER:
Output:
```
[
 ...
 {
  "name": "show_playlist_library",
  "description": "Search or show a list of playlists in your playlist library."
 },
 {
  "name": "show_playlist",
  "description": "Get detailed information about a specific playlist. You can view your own playlists or others' playlists if they are public."
 },
 ...
]
```

ASSISTANT:
The `show_playlist_library` API seems to be the one we need. Let's check its specifications.

```python
```

```
print(apis.api_docs.show_api_doc(app_name='spotify', api_name='show_playlist_library'))
```

USER:
Output:
```
{
 "app_name": "spotify",
 "api_name": "show_playlist_library",
 "path": "/spotify/library/playlists",
 "method": "GET",
 "description": "Search or show a list of playlists in your playlist library.",
 "parameters": [
  {
   "name": "access_token",
   "type": "string",
   "required": true,
   "description": "Access token obtained from spotify app login.",
   "default": null,
   "constraints": []
  },
  {
   "name": "query",
   "type": "string",
   "required": false,
   "description": "The search query string.",
   "default": "",
   "constraints": []
  },
  {
   "name": "is_public",
   "type": "boolean",
   "required": false,
   "description": "Whether to show public playlists or private playlists.",
   "default": null,
   "constraints": []
  },
  {
   "name": "page_index",
   "type": "integer",
   "required": false,
   "description": "The index of the page to return.",
   "default": 0,
   "constraints": [
    "value >= 0"
   ]
  },
  {
   "name": "page_limit",
   "type": "integer",
   "required": false,
   "description": "The maximum number of results to return per page.",
   "default": 5,
   "constraints": [
    "value >= 1, <= 20"
   ]
  },
  {
   "name": "sort_by",
   "type": "string",
   "required": false,
   "description": "The attribute to sort the library playlists by prefixed with +/- to
↪ reflect ascending/descending. Valid attributes: created_at and title. ...",
   "default": null,
   "constraints": []
  }
 ],
 "response_schemas": {
  "success": [
   {
    "playlist_id": 1,
    "title": "string",
    "is_public": true,
    "rating": 0.0,
    "like_count": 1,
    "review_count": 1,
    "owner": {
     "name": "string",
     "email": "user@example.com"
```

```
    },
    "created_at": "2019-01-01T00:00:00",
    "song_ids": [
     1
    ]
   }
  ],
  "failure": {
   "message": "string"
  }
 }
}
```

ASSISTANT:
I need to page through all the playlists to get the list of playlists and save it in `playlists`.

```python
page_index = 0
playlists = []
while page_index < 10:
  playlist_page = apis.spotify.show_playlist_library(access_token=spotify_access_token, page_index=page_index)
  if playlist_page:
    playlists.extend(playlist_page)
    page_index += 1
  else:
    break
num_playlists = len(playlists)
print(num_playlists)
```

USER:
Output:
```
23
```

ASSISTANT:
Now that the task is completed, I can call apis.supervisor.complete_task(). Since this task has an answer to be returned, I will pass that as an argument.

```python
apis.supervisor.complete_task(answer=num_playlists)
```

USER:
Output:
Marked the active task complete.

USER:
Key instructions:

A. General instructions:

- Act fully on your own. You must make all decisions yourself and never ask me or anyone else to confirm or clarify. Your role is to solve the task, not to bounce questions back, or provide me directions to follow.
- You have full access -- complete permission to operate across my connected accounts and services.
- Never invent or guess values. For example, if I ask you to play a song, do not assume the ID is 123. Instead, look it up properly through the right API.
- Never leave placeholders; don't output things like "your_username". Always fill in the real value by retrieving it via APIs (e.g., Supervisor app for credentials).
- When I omit details, choose any valid value. For example, if I ask you to buy something but don't specify which payment card to use, you may pick any one of my available cards.
- Avoid collateral damage. Only perform what I explicitly ask for. Example: if I ask you to buy something, do not delete emails, return the order, or perform unrelated account operations.

B. App-specific instructions:

- All my personal information (biographical details, credentials, addresses, cards) is stored in the Supervisor app, accessible via its APIs.
- Any reference to my friends, family or any other person or relation refers to the people in my phone's contacts list.
- Always obtain the current date or time, from Python function calls like `datetime.now()`, or from the phone app's get_current_date_and_time API, never from your internal clock.

```
- All requests are concerning a single, default (no) time zone.
- For temporal requests, use proper time boundaries, e.g., when asked about periods like "
↪ yesterday", use complete ranges: 00:00:00 to 23:59:59.
- References to "file system" mean the file system app, not the machine's OS. Do not use OS
↪  modules or functions.
- Paginated APIs: Always process all results, looping through the page_index. Don't stop at
↪  the first page.

C. Code-operation instructions

- Make sure to end code blocks with ``` followed by a newline(
).
- Remember, you can use the variables in your code in subsequent code blocks.
- Remember that the email addresses, access tokens and variables (e.g. spotify_password) in
↪  the example above are not valid anymore.
- Always look at API specifications (using apis.api_docs.show_api_doc) before calling an
↪ API.
- Write small chunks of code and only one chunk of code in every step. Make sure everything
↪  is working correctly before making any irreversible changes.
- The Python environment supports the standard library. But system-level operations that
↪ may access or affect OS files, processes, etc., are not allowed and will raise an error
↪ if called.
- To interact with apps, only use the provided app APIs, and not the corresponding Python
↪ packages, e.g., do NOT use `spotipy` for Spotify.
- The provided API documentation has both the input arguments and the output JSON format.
↪ Use this information when making API calls and parsing their outputs.

D. Task-completion instructions:

You must call the `apis.supervisor.complete_task` API after completing the task.
- If an answer is needed, e.g., for "How many songs are in the Spotify queue?", call it
↪ with the appropriate answer argument value.
- If no answer is required, e.g., for "Start my Spotify music player.", omit the answer
↪ argument (or set it to None/null).
- The task is doable, but if you cannot find a way, you can call it with status="fail" to
↪ exit with failure.

When the answer is given:
- Keep answers minimal. Return only the entity, number, or direct value requested - not
↪ full sentences.
  E.g., for the song title of the current playing track, return just the title.
- Numbers must be numeric and not in words.
  E.g., for the number of songs in the queue, return "10", not "ten".

The task you must solve will be given to you in the next message.
```

AppWorld — fixed assistant acknowledgement

```
Ok.
```

D.5 τ^2-BENCH

τ^2's standing instruction is not a fixed string: it is assembled per domain at server start-up and fetched by the client over `/system_prompt`, so a mixed-domain batch carries a different policy and a different tool list per task. Three slots are filled in: `{policy}` is the domain's policy document, shipped with τ^2-bench and inserted verbatim; `{tools}` is that domain's tool list rendered as signatures; `{ending}` selects one of the two closing blocks below (`TAU2_PROMPT_VARIANT`, default `strict`).

τ^2 — standing instruction, ReAct variant (template)

```
You are a customer service agent. You talk to a customer and use tools to act on their
↪ behalf, and you must follow the policy below exactly.

# Policy

{policy}

# Tools

{tools}
```

```
# Response format

EVERY turn you MUST reply with exactly a Thought and an Action:

Thought:
<your reasoning about what to do next>

Action:
<a single tool call, OR a message to the customer>

The Action is interpreted as follows:
- If it looks like `tool_name(arg='value', other=123)` it is executed as a tool call,
  and the next observation is the tool's result.
- Otherwise it is sent verbatim to the customer as your message, and the next
  observation is their reply.

Rules for the Action line:
- Write the tool call on ONE line. Use only the parameters listed above.
- Never put a tool call and a message in the same Action -- do one or the other.
- Never wrap the Action in quotes, backticks or a code block.
{ending}
```

The `{ending}` slot takes one of:

τ^2 — ending `strict` (default)

```
- The customer ends the conversation when they are satisfied. Do not end it yourself
  while anything they asked for is still unresolved.

# Before you act

- Authenticate FIRST. You cannot use a user id until a tool returned it to you. Call
  find_user_id_by_email, or find_user_id_by_name_zip, and use the id it returns.
  Never invent, guess, or reuse an id from an example.
- Never call transfer_to_human_agents unless the customer explicitly asks for a human.
  Being unsure, missing a detail, or finding the task hard is NOT a reason to transfer
  -- ask the customer for what you are missing, or look it up with a tool. A transfer
  leaves the task unfinished and scores zero.
- Finish the job. Reading with get_* tools changes nothing on its own; almost every
  request needs a write tool (cancel_, modify_, return_, exchange_) before it is done.
  Confirm the details with the customer, then make that call.
- Only call `done()` after the write tool has succeeded, or after the customer has
  confirmed they need nothing else.
```

τ^2 — ending `base`

```
- The customer ends the conversation when they are satisfied. Only call `done()`
  yourself once everything the customer asked for is resolved or you have told them
  you cannot help -- calling it early ends the episode and scores zero.
```

The conversation is laid out as for the other environments, with the assembled instruction as the first user turn:

τ^2 — conversation layout, ReAct variant

```
[USER]
<the assembled standing instruction above, for this task's domain>

[ASSISTANT]
Ok.

{history}

[USER]
{observation}

[ASSISTANT]
<- Thought + Action; the Action is a tool call or a message to the customer
```

D.5.1 NATIVE VARIANT

Under `TAU2_PROMPT_VARIANT=native` the policy speaks τ^2's own protocol instead of the ReAct wrapper: the instruction becomes a *system* message with no acknowledgement turn, the response-format block and the extra rules are dropped, and the domain tools are written out exactly as the chat template renders a native `tools=` list, so the model emits `<tool_call>` JSON.

τ^2 — standing instruction, native variant

```
You are a customer service agent that helps the user according to the policy provided below
↪ .
In each turn you can either:
- Send a message to the user.
- Make a tool call.
You cannot do both at the same time.

Try to be helpful and always follow the policy.

<policy>
{policy}
</policy>

# Tools

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

You are provided with function signatures within <tools></tools> XML tags:
<tools>

</tools>

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

where each line inside `<tools>` is one tool's OpenAI JSON schema.

D.5.2 WHAT FILLS {POLICY} AND {TOOLS}

The policy documents come from τ^2-bench itself and are inserted unmodified; measured on the domains we train and evaluate on:

domain	policy (characters)	policy (words)	tools advertised
retail	6,699	1,158	16
airline	7,676	1,313	15
telecom	23,318	3,715	14

The tool list is rendered one tool per entry as `- name(arg: type, optional: type = None)` followed by the tool's own description, indented. For `retail` this block is 3,905 characters; it is reproduced here in full as the concrete example of the `{tools}` slot.

τ^2 — {tools} rendered for the retail domain

```
- calculate(expression: string)
    Calculate the result of a mathematical expression.
- cancel_pending_order(order_id: string, reason: string)
    Cancel a pending order. If the order is already processed or delivered,

it cannot be cancelled. The agent needs to explain the cancellation detail
and ask for explicit user confirmation (yes/no) to proceed. If the user confirms,
the order status will be changed to 'cancelled' and the payment will be refunded.
The refund will be added to the user's gift card balance immediately if the payment
was made using a gift card, otherwise the refund would take 5-7 business days to process.
The function returns the order details after the cancellation.
- done()
    Call this function when you are done with the task.
```

```
- exchange_delivered_order_items(order_id: string, item_ids: array, new_item_ids: array,
↪ payment_method_id: string)
    Exchange items in a delivered order to new items of the same product type.

For a delivered order, return or exchange can be only done once by the agent.
The agent needs to explain the exchange detail and ask for explicit user confirmation (yes/
↪ no) to proceed.
- find_user_id_by_email(email: string)
    Find user id by email. If the user is not found, the function will return an error
↪ message.
- find_user_id_by_name_zip(first_name: string, last_name: string, zip: string)
    Find user id by first name, last name, and zip code. If the user is not found, the
↪ function

will return an error message. By default, find user id by email, and only call this
↪ function
if the user is not found by email or cannot remember email.
- get_order_details(order_id: string)
    Get the status and details of an order.
- get_product_details(product_id: string)
    Get the inventory details of a product.
- get_user_details(user_id: string)
    Get the details of a user, including their orders.
- list_all_product_types()
    List the name and product id of all product types.

Each product type has a variety of different items with unique item ids and options.
There are only 50 product types in the store.
- modify_pending_order_address(order_id: string, address1: string, address2: string, city:
↪ string, state: string, country: string, zip: string)
    Modify the shipping address of a pending order. The agent needs to explain the
↪ modification detail and ask for explicit user confirmation (yes/no) to proceed.
- modify_pending_order_items(order_id: string, item_ids: array, new_item_ids: array,
↪ payment_method_id: string)
    Modify items in a pending order to new items of the same product type. For a pending
↪ order, this function can only be called once. The agent needs to explain the exchange
↪ detail and ask for explicit user confirmation (yes/no) to proceed.
- modify_pending_order_payment(order_id: string, payment_method_id: string)
    Modify the payment method of a pending order. The agent needs to explain the
↪ modification detail and ask for explicit user confirmation (yes/no) to proceed.
- modify_user_address(user_id: string, address1: string, address2: string, city: string,
↪ state: string, country: string, zip: string)
    Modify the default address of a user. The agent needs to explain the modification
↪ detail and ask for explicit user confirmation (yes/no) to proceed.
- return_delivered_order_items(order_id: string, item_ids: array, payment_method_id: string
↪ )
    Return some items of a delivered order.

The order status will be changed to 'return requested'.
The agent needs to explain the return detail and ask for explicit user confirmation (yes/no
↪ ) to proceed.
The user will receive follow-up email for how and where to return the item.
- transfer_to_human_agents(summary: string)
    Transfer the user to a human agent, with a summary of the user's issue.

Only transfer if
 -  the user explicitly asks for a human agent
 -  given the policy and the available tools, you cannot solve the user's issue.
```

D.6 Auxiliary Prompts (Action Forecasting)

The forecasting objective is trained on a separate sequence, not on the episode itself: the dialogue is truncated after an observation and one synthetic user turn is appended, with the realised next-K actions as the target (one per line, no other text). `{k}` is the horizon; `{history}` is again the dialogue up to that observation.

Action-forecast sequence

```
[USER]
<the environment's standing instruction, as above>

[ASSISTANT]
<the fixed acknowledgement, as above>
```

```
{history}

[USER]
{observation}

[USER]
Plan ahead: list the next {k} actions you will take to make progress on the task, starting
↪ with the action you take right now, one action per line.

[ASSISTANT]
<- target: the K actions the episode actually executed
```

A sub-goal variant of the target is also implemented and is disabled in the reported runs:

Sub-goal target variant (disabled)

```
List the next {k} sub-goals you will actually accomplish from here -- the milestones you
↪ will complete -- one per line.
```