The Road Not Taken: Hindsight Exploration for LLMs in Multi-Turn RL

Huaxiaoyue Wang
Cornell University
yukiwang@cs.cornell.edu

Sanjiban Choudhury
Cornell University
sanjibanc@cornell.edu

Abstract

Multi-turn reinforcement learning provides a principled framework for training LLM agents, but exploration remains a key bottleneck. Classical exploration strategies such as ϵ-greedy and upper confidence bounds select random actions, failing to efficiently explore the combinatorial space of multi-turn token sequences. Our key insight is that LLMs can use hindsight to guide exploration: by analyzing completed trajectories and proposing counterfactual actions that could have led to higher returns. We propose HOPE (**H**indsight **O**ff-**P**olicy **E**xploration), which integrates hindsight-guided exploration into both the actor and critic stages of multi-turn RL. HOPE improves the critic's state-action coverage by generating rollouts from counterfactual actions, and steers the actor's exploration in RL by using a learned counterfactual generator to propose alternative actions. Experimental results show that HOPE outperforms strong multi-turn RL baselines in task-oriented dialogue tasks, TwentyQuestions (success: $0.82 \rightarrow 0.97$), GuessMyCity (success: $0.68 \rightarrow 0.75$), and tool-use dialogue task CarDealer (success: $0.72 \rightarrow 0.77$). Our code is available at https://portal-cornell.github.io/HOPE/

1 Introduction

Reinforcement learning (RL) has proven effective for aligning large language models (LLMs) with human preferences [31, 36, 41] and enhancing their reasoning capabilities [13, 24, 45, 48]. Recent works apply RL to train LLM agents for multi-turn decision-making tasks, such as code generation [18, 23], web navigation [4, 32, 33], and task-oriented dialogues [1, 46, 55], where each action by the agent influences future states in the environment.

However, exploration remains a key challenge in RL [6, 42, 44]. Classical exploration strategies that sample random actions, such as ϵ-greedy and upper confidence bound [3], fail to efficiently explore the expansive action space of LLM agents, where actions are token sequences spanning multiple turns. Random perturbations rarely produce coherent alternative actions, making it difficult to explore the space of multi-turn sequences in a purposeful way.

Our key insight is that LLMs can guide exploration through hindsight reasoning—by analyzing completed trajectories and proposing counterfactual actions that might have led to higher returns. Prior works empirically show that LLMs exhibit this form of hindsight reasoning [5, 19, 29, 39], often revising their responses based on feedback. However, naively imitating these counterfactual actions does not guarantee policy improvement—they may be suboptimal or unrealizable.

We present HOPE (**H**indsight **O**ff-**P**olicy **E**xploration), which leverages environment rewards and a learned critic to identify the counterfactual actions that have resulted in improved outcomes. HOPE integrates hindsight-guided exploration into both stages of actor-critic RL. In critic training, it augments the replay buffer by splicing in high-value counterfactual actions at intermediate timesteps of past trajectories and then rolling out the current policy, thereby broadening state–action coverage.

First Exploration in AI Today Workshop at ICML (EXAIT at ICML 2025).

In actor updates, it biases exploration toward promising regions by sampling candidate actions from a learned counterfactual generator and evaluating them via the critic. We show that HOPE is both easy to incorporate into existing multi-turn RL pipelines and closely aligned with the principles of posterior sampling, as it explores by generating actions conditioned on past outcomes.

Our key contributions are:

1. A novel framework, HOPE, that injects hindsight-guided exploration into both critic and actor stages of multi-turn RL.
2. A counterfactual action generator that steers the actor's exploration during RL by proposing alternative actions.
3. Experiments validation on diverse multi-turn decision-making domains [1], showing that HOPE outperforms leading multi-turn RL approaches on task-oriented dialogue and tool-use tasks.

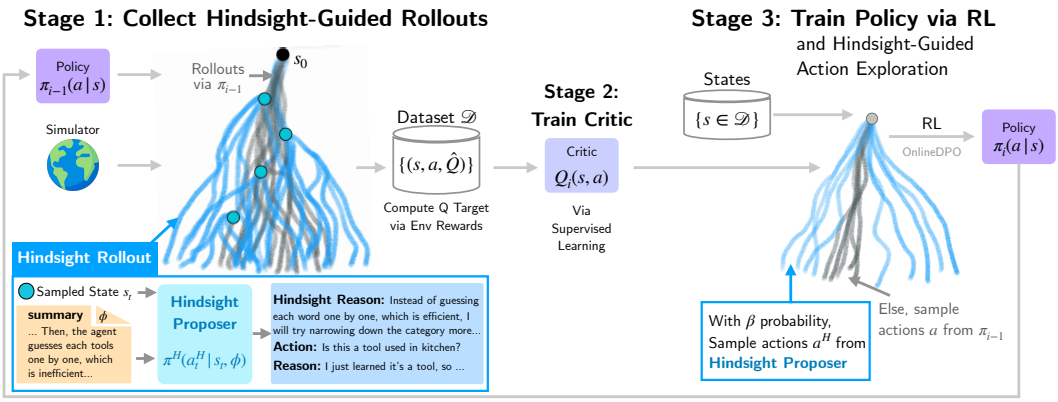


Figure 1: **Overview of** HOPE that integrates hindsight-guided exploration in actor-critic RL training. The training loop is iterative. Stage 1: Collect diverse rollouts to compute Q targets by augmenting past rollouts with counterfactual actions and rolling out the remaining steps with the current policy π_i. Stage 2: Train the critic $Q_i(s, a)$ via supervised learning. Stage 3: Train the policy via RL and hindsight guided action exploration, where it samples counterfactual actions with probability β.

2 Preliminaries

Problem Formulation. We model the multi-turn decision-making problem as a Markov Decision Process (MDP) for the LLM agent. At each turn t, the agent receives state $s_t = \{o_0, a_0, \ldots, o_t\}$, the history of observations and actions so far. It then takes an action a_t, a token sequence, and transitions to a new state s_{t+1} according to the environment's transition dynamics. We assume access to a sparse reward signal $r(s_t, a_t)$ only at the end of the trajectory. The goal is to learn a policy $\pi(a_t \mid s_t)$ that maximizes the expected discounted return: $\mathbb{E}_\pi \left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)\right]$.

Actor Critic for Multi-Turn RL. Multi-turn RL can be viewed as a hierarchical RL problem consisting of two nested RL loops: (1) turn-level RL and (2) token-level RL. We solve the turn-level RL problem by rolling out the agent in the environment, collecting state-action trajectories and training a Process Reward Model (PRM) $Q(s, a)$, akin to a critic in actor-critic RL. The LLM policy is then solved by optimizing the PRM using a token-level RL algorithm. While this recipe has been used to great success in math and reasoning settings [24, 38, 45, 48], this framework is underexplored in the LLM agent setting.

The training loop is iterative: at each iteration i, the current policy π_{i-1} is rolled out to generate trajectories. For each encountered state-action pair (s, a), all trajectories passing through (s, a) are stored in a replay buffer $\mathcal{G}(s, a)$. The PRM target dataset $\mathcal{D} = \{(s, a, \hat{Q})\}$ is constructed using Monte Carlo estimates:

$$\hat{Q}(s, a) = \frac{1}{|\mathcal{G}(s, a)|} \sum_{(s_t, a_t) \in \mathcal{G}(s,a)} \sum_{k=t}^{T-1} \gamma^{k-t} r(s_k, a_k). \tag{1}$$

The PRM Q_i is trained on $\mathcal{D}$, and the policy π_i is then trained to maximize Q_i while staying close to π_{i-1}, using algorithms such as PPO [37], rejection sampling [15], or Online DPO [16].

The effectiveness of this framework hinges on the quality and diversity of trajectories in the replay buffer. When exploration is limited, the buffer contains few high-reward state-action pairs, leading to poor value estimates and stagnated policy improvement.

Algorithm 1 Actor-Critic RL with HOPE (Hindsight Off-Policy Exploration)

Initialize with agent policy π_0
for iteration $i = 1, \ldots, K$ **do**
  // Stage 1: Collect rollouts (Explore with HOPE)
  Collect on-policy rollouts $\{(\ldots, s_t, a_t, r_t, \ldots)\}$ using π_{i-1} and store in replay buffer $\mathcal{G}(s, a)$
  Collect hindsight-guided rollouts Algorithm 2 and maintain data ratio α in $\mathcal{G}(s, a)$
  // Stage 2: Train Process Reward Model
  Compute PRM targets via (1) and aggregate into dataset $\mathcal{D} = \{(s, a, \hat{Q})\}$
  Train PRM Q_i to minimize soft binary cross-entropy loss:

$$Q_i = \arg\min_Q -\mathbb{E}_{(s,a,\hat{Q})\sim\mathcal{D}} \left[\hat{Q} \log Q(s, a) + (1 - \hat{Q}) \log(1 - Q(s, a))\right] \tag{2}$$

  // Stage 3: Train Policy via RL (Explore with HOPE)
  Train policy π_i to maximize Q_i, **exploring with probability β via hindsight proposer π^H**

$$\pi_i = \arg\max_\pi \mathbb{E}_{s\sim\mathcal{D}, a\sim\text{Sample}(s,\pi_i,\pi^H,\beta)} [Q_i(s, a)] - \beta \mathcal{D}_{\text{KL}} [\pi(a \mid s) \| \pi_{i-1}(a \mid s)] \tag{3}$$

end for
return Best $\pi \in \{\pi_1, \ldots, \pi_K\}$ on validation dataset

3 Approach

We introduce HOPE (**H**indsight **O**ff-**P**olicy **E**xploration) in Algorithm 1, a framework that leverages LLMs' hindsight reasoning to guide exploration in multi-turn RL. Unlike conventional approaches that rely on random action noise, HOPE proposes counterfactual actions by analyzing completed trajectories. A hindsight proposer generates outcome-conditioned alternatives from trajectory summaries (Section 3.1), which are then used to improve state-action coverage during critic training (Section 3.2) and steer exploration during actor training (Section 3.3).

Algorithm 2 HOPE Data Collection For Critic

Input: Dataset of onpolicy rollouts $\{\tau\}$; current policy π; rollout summarizer $\pi^{\text{sum}}(\phi \mid \tau)$; hindsight proposer $\pi^H(a^H \mid s_t, \phi)$.
for each onpolicy rollout τ **do**
  Get summary $\phi \sim \pi^{\text{sum}}(\cdot \mid \tau)$
  for each randomly sampled timestep t **do**
    Get counterfactual $a_t^H \sim \pi^H(\cdot \mid s_t, \phi)$
    Create partial rollout $\tau^H = (\ldots, s_t, a_t^H)$
    Complete τ^H with π and store in dictionary $\mathcal{G}^H(s, a)$
  end for
end for
return State-action dictionary $\mathcal{G}^H(s, a)$

3.1 Hindsight Proposer

We introduce a *hindsight proposer*, π^H, a LLM policy that generates a counterfactual action given a state s_t and a completed trajectory $\tau = (s_0, a_0, r_0, \ldots)$. The goal is to produce an alternative action that might have led to a better outcome.

A key challenge is that completed trajectories are long and entangled: they contain many state-action pairs whose contributions to final reward are hard to disentangle. Prompting an LLM directly on τ often yields shallow or noisy revisions. To address this, we introduce a two-step decomposition. First, we use a summarizer LLM π^{sum} to generate a compact, natural language summary $\phi \sim \pi^{\text{sum}}(\cdot \mid \tau)$ that reflects the policy's high-level strategy and assesses the effectiveness of individual actions. This summary typically includes both behavioral intent (e.g., "the agent fails to consider a common category...") and retrospective feedback ("this prevents the agent from searching effectively..."). Then,

conditioned on the state s_t and summary ϕ, the hindsight proposer π^H generates a counterfactual action $a_t^H \sim \pi^H(\cdot \mid s_t, \phi)$.

Prior work has demonstrated that LLMs are capable of hindsight reasoning—either by refining their responses based on feedback [5, 19, 29, 39] or generating post-hoc explanations conditioned on ground-truth answers [47, 50, 52]. However, these capabilities have primarily been applied to improve generation quality or interpretability. In contrast, we use hindsight reasoning to guide exploration: the LLM summarizes completed trajectories and proposes counterfactual actions that could have led to better outcomes. This allows the agent to incorporate structured exploratory data during training, rather than relying solely on stochastic or undirected exploration strategies.

3.2 Hindsight-Guided Critic Training

Algorithm 2 presents hindsight-guided data collection to expand state-action coverage in the critic training data. We first collect a small set of on-policy trajectories $\{\tau\}$ with the current policy π_{i-1}. We summarize each trajectory τ as ϕ and randomly select a state s_t to query the hindsight proposer for a counterfactual action $a_t^H \sim \pi^H(\cdot \mid s_t, \phi)$. To evaluate whether a_t^H can improve outcomes, we augment the original trajectory with a_t^H before completing the rollout with π_{i-1} and obtaining $\tau^H = (\ldots, s_t, a_t^H, r_t^H, s_{t+1}, a_{t+1}, \ldots)$.

Then, we store all collected trajectories that pass through each encountered state-action pair (s, a) in a replay buffer $\mathcal{G}(s, a)$ before computing target Q-values via Monte Carlo Estimation. To account for off-policy data introduced by the hindsight proposer, we construct $\mathcal{G}(s, a)$ using a mixture that consists α percentage of hindsight data and $(1 - \alpha)$ data from on-policy trajectories $\{\tau\}$. We empirically investigate the effect of ratio α in Section D.1.

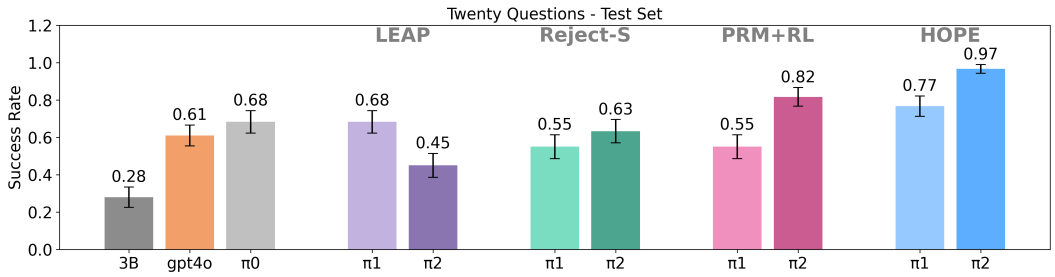
3.3 Hindsight-Guided Actor Training

To guide action exploration during RL, `HOPE` samples counterfactual actions from the hindsight proposer $a^H \sim \pi^H(s, \phi)$ with probability β given a state s and a corresponding summary ϕ. Setting $\beta = 0$ recovers the original actor objective where actions are only sampled from the LLM agent. The critic $Q(s, a)$ provides supervision on the counterfactual actions, allowing the actor to learn from actions that lead to higher Q-estimates. Because counterfactual actions are explicitly generated to differ from collected experiences, they help overcome premature convergence when actions sampled from the policy are clustered around local optima [8, 49, 54].

3.4 Connection to Posterior Sampling

Our method draws inspiration from posterior sampling (PS) which is a well-studied exploration strategy in multi-armed bandit [9, 43] and RL [17, 42]. In the context of RL, the algorithm maintains and samples from a posterior over MDPs M given the history of all trajectories collected so far: $P(M|\mathcal{H})$. In every episode i, it samples a new MDP $M_i \sim P(M \mid \mathcal{H})$, computes the optimal policy π_i, rolls it out to collect interaction data and adds it to $\mathcal{H}$, and updates the posterior. PS is provably exploration-efficient in terms of Bayesian regret, achieving near-optimal bounds in tabular settings [17]. The key intuition is that posterior sampling balances exploration and exploitation by implicitly encouraging optimism through sampling: sampled MDPs often favor under-explored regions of the environment due to posterior uncertainty. While elegant in theory, extending PS to large-scale or function-approximation settings introduces practical challenges, such as maintaining a posterior over high-dimensional model classes and solving for optimal policies in sampled MDPs at each episode.

In contrast to classical PS, we make a number of practical approximations. Rather than explicitly maintaining and updating a full posterior over MDPs, we instead generate counterfactual actions from a hindsight policy $\pi^H(s, \phi)$ conditioned on a trajectory summary ϕ. This hindsight generator implicitly captures posterior over plausible MDPs in its reasoning, and then samples alternative actions that would be optimal under the MDP. Unlike PS, which updates its posterior using the full interaction history $\mathcal{H}$, our approach uses only a compact trajectory-level summary ϕ. This simplification sacrifices the formal guarantees of posterior sampling, since the hindsight policy may not reflect a true posterior update. However, when the LLM prior is well-calibrated with the MDP distribution, this approximation can yield effective exploration without the computational burden of maintaining and updating an explicit posterior.



		TwentyQuestions		GuessMyCity		CarDealer (Tool)	
		Return ↑	Success ↑	Return ↑	Success ↑	Return ↑	Success ↑
	gpt4o	1.00 (0.11)	0.61 (0.06)	**1.00 (0.12)**	0.56 (0.05)	1.00 (0.09)	0.59 (0.05)
	3B	0.09 (0.04)	0.28 (0.05)	0.01 (0.01)	0.01 (0.01)	0.40 (0.07)	0.51 (0.05)
	π_0	1.08 (0.11)	0.67 (0.05)	0.80 (0.09)	0.70 (0.05)	1.03 (0.08)	0.66 (0.05)
LEAP	π_1	1.16 (0.12)	0.68 (0.06)	0.35 (0.04)	0.59 (0.05)	0.77 (0.08)	0.52 (0.05)
	π_2	0.52 (0.09)	0.45 (0.06)	0.29 (0.03)	0.62 (0.05)	0.86 (0.08)	0.57 (0.05)
Reject-S	π_1	0.63 (0.11)	0.55 (0.06)	0.46 (0.02)	0.69 (0.02)	1.07 (0.08)	0.68 (0.05)
	π_2	0.90 (0.13)	0.63 (0.06)	0.31 (0.03)	0.62 (0.05)	1.00 (0.08)	0.62 (0.05)
PRM+RL	π_1	1.19 (0.16)	0.55 (0.06)	0.35 (0.05)	0.66 (0.05)	1.00 (0.08)	0.67 (0.05)
	π_2	1.05 (0.09)	0.82 (0.05)	0.54 (0.03)	0.66 (0.02)	1.14 (0.08)	0.72 (0.04)
HOPE	π_1	1.15 (0.13)	0.77 (0.05)	0.45 (0.05)	0.74 (0.05)	1.07 (0.08)	0.69 (0.05)
	π_2	**1.91 (0.09)**	**0.97 (0.02)**	0.91 (0.10)	**0.77 (0.04)**	**1.20 (0.08)**	**0.77 (0.04)**

Table 1: **Policy performance on test sets of three task-oriented conversational environments.** We report the average normalized return and success rate with standard error, and we highlight the top-performing approaches. The return is normalized with respect to GPT-4o's performance.

4 Experiments

Our experiments aim to evaluate the effectiveness of HOPE in multi-turn conversational domains with sparse environment rewards. Below, we summarize the key research questions we investigated along with our main findings:

1. **Does HOPE outperform state-of-the-art IL and RL algorithms? (Section 4.2)** As shown in Table 1, HOPE achieves the highest success rates, improving over baselines across all domains, with gains in TwentyQuestions (0.82 → 0.97), GuessMyCity (0.68 → 0.75), and CarDealer (0.72 → 0.77).
2. **How effective is hindsight-guided data collection for critic training? (Section 4.3)** Figure 2 shows that HOPE achieves a success rate of 0.77 on TwentyQuestions, outperforming both the unguided H-Temp (0.55) and Explore-Prompt (0.72) exploration strategies.
3. **Does hindsight-guided exploration for RL training improve performance? (Section 4.4)** Table 2 shows that hindsight-guided exploration improves success rates in the TwentyQuestions environment, especially when the critic data collection is unguided (0.55 → 0.62).

4.1 Setup

Multi-Turn Environments. We evaluate our approach across 3 multi-turn conversational environments proposed in the LMRL Gym benchmark [1]. Each environment models a conversation, where the LLM agent must talk with a simulated user to achieve some goal. Below, we describe the conversation structure, reward function, success criteria, and dataset splits for each environment:

- **TwentyQuestions.** In this environment, the LLM agent must identify a secret object selected by the simulator. To do so, the agent has up to 20 turns, each consisting of a single yes-or-no question aimed at narrowing down the possibilities. The answers are simulated by prompting a Llama-3.2-3B-Instruct model [15] with the secret object name and the question. The LLM agent

receives a -1 reward on each turn, and the episode ends with a reward of 0 when the secret object is correctly guessed. The train and validation set share the same object categories but contain different objects. The test set contains new objects from categories not seen in the train/validation sets.

- **GuessMyCity.** As in the previous environment, the simulator selects a secret city that the LLM agent must identify. However, instead of asking yes-or-no questions, the agent can ask up to 10 open-ended questions, each eliciting a free-form response. We use Llama-3.2-3B-Instruct [15] to simulate the environment. The episode ends successfully with reward of 0 when the agent correctly guesses the secret city, accruing a reward of -1 for all previous turns. The training and validation sets include cities from the same set of countries, while the test set uses an unseen set of countries.
- **CarDealer (Tool).** We extend the original Car Dealer environment to incorporate tool use. In this environment, the LLM agent plays the role of a car dealer aiming to sell a vehicle to a buyer within 10 dialogue turns. At each turn, the agent may first issue a database tool call to search the inventory and then respond to the buyer. The interaction ends when the buyer agrees to a purchase, with a reward of $(\texttt{purchase\_cost})^2/(\texttt{budget} \times \texttt{market\_price})$; all prior turns yield zero reward. We simulate buyers with diverse personalities and constraints using Qwen2.5-14B-Instruct [35]. The validation set models the same buyer profiles as the training set, while varying the available cars. The test set has both unseen buyer profiles and cars.

Baselines. We evaluate the effectiveness of HOPE against a range of baseline approaches. We begin by benchmarking the zero-shot performance of GPT-4o and the base model, Llama-3.2-3B-Instruct [15] (denoted as **3B**). Following RLHF [31], we supervised finetune the base model for 3 epochs using gpt4o's rollouts on the train set and denote this policy as π_0. Using π_0 as the starting model for training, we compare various imitation learning (IL) and reinforcement learning (RL) methods. LEAP [11] is an IL approach that directly finetunes the policy to imitate corrective actions from a teacher policy, which is the hindsight proposer for our setting (Sec. 3.1). For RL, we compare with Rejection Sampling (Reject-S for short) [15] that skips training a critic. It iteratively finetunes the policy on successful trajectories as indicated by the environment sparse reward. Meanwhile, PRM+RL is an actor-critic method that explicitly trains a critic similar to HOPE, but it does not leverage the hindsight proposer for its critic's data collection (i.e, $\alpha = 0$) and simply samples actions from the current policy with high temperature 1.0.

Training Details. Appendix includes detailed information about the ratio α of hindsight data used to train HOPE's critic in each domain, prompts, and training hyperparameters,

Metrics. We report the **success rate** and normalized **returns** (i.e., cumulative rewards normalized with respect to GPT-4o's performance) on the test set. We evaluate all intermediate training checkpoints on the validation set and select the best-performing one for final evaluation on the test set.

4.2 Does HOPE outperform state-of-the-art IL and RL algorithms?

Table 1 shows that HOPE consistently outperforms all baselines, achieving the highest rewards and success rates across all domains. Among the RL baselines, Reject-S often fails to improve over the initial policy π_0, particularly in the TwentyQuestions and CarDealer environments. This is because it only updates the policy using state-action pairs from successful trajectories, ignoring potentially informative rollouts that result in failure. In contrast, PRM+RL is more competitive, as its actor is trained to maximize Q-values estimated by a critic, enabling it to learn from both successful and unsuccessful trajectories. However, since its critic is trained only on on-policy data, it suffers from limited state-action coverage. This limits the actor's ability to discover higher-value behaviors. HOPE overcomes this limitation through hindsight-guided exploration, which augments the critic training data with diverse, high-value counterfactual actions and improves critic supervision. Fi-

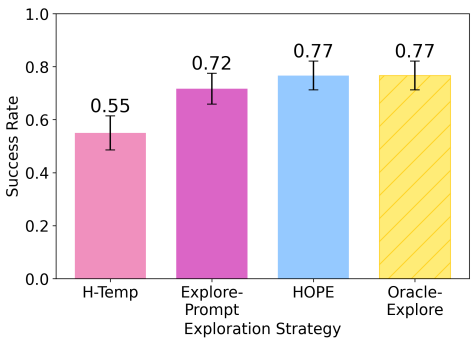


Figure 2: **Policy performance given different exploration strategies.** We use the test set of TwentyQuestions to evaluate π_1 trained with different exploration approaches and report the average success with standard error.

nally, the IL baseline LEAP often underperforms, as it lacks access to reward signals and simply imitates the hindsight proposer, which may generate suboptimal actions in certain contexts.

4.3 How effective is hindsight-guided data collection for critic training?

We evaluate the effectiveness of hindsight-guided exploration against both unguided and guided exploration strategies. For each method, we train a policy π_1 using actor-critic RL and report its test-time performance on the TwentyQuestions domain.

As an unguided baseline, `H-Temp` (PRM+RL) explores by sampling actions from π_0 with a high temperature of 1.0, collecting 30 rollouts per task to train the critic. We also compare against two guided strategies, `Explore-Prompt` and `Oracle-Explore`, both of which follow the HOPE protocol in Algorithm 2 by collecting 14 on-policy rollouts with π_0, followed by 16 guided rollouts. Instead of using a hindsight proposer, `Explore-Prompt` samples 5 actions from π_0 (at temperature 1.0) and then augments the prompt to explicitly request actions that differ from those samples. `Oracle-Explore`, in contrast, uses an oracle proposer—specifically, the final HOPE policy π_2—to generate actions, serving as an upper bound for guided exploration.

4.4 Does hindsight-guided exploration for RL training improve performance?

We investigate how hindsight-guided exploration influences actor training by varying the hyperparameter β that controls the probability of sampling counterfactual actions from the hindsight generator.

Hindsight	Critic	x	x	✓	✓
Explore in	Actor	x	✓	x	✓
success		0.55 (0.06)	0.62 (0.06)	0.77 (0.05)	0.77 (0.05)

Table 2: **Effect of hindsight-guided exploration during actor training.** ✓ in Critic represents $\alpha = 0.4$ where the critic is trained with 40% hindsight data. ✓ in Actor represents $\beta = 0.5$. We report the average success rate and standard error in TwentyQuestions test set.

Setup. We compare four configurations: $(\alpha = 0.0, \beta = 0.0)$, which does not utilize any hindsight exploration; $(\alpha = 0.0, \beta = 0.5)$, which only uses hindsight-guided exploration during actor training; $(\alpha = 0.4, \beta = 0.0)$, which only uses hindsight-guided critic trained on 40% hindsight data; $(\alpha = 0.4, \beta = 0.5)$, which uses hindsight exploration both in critic and actor training. To cost-efficiently generate counterfactual actions during RL training, we distill the GPT-4o hindsight proposer π^H into a Llama-3.2-3B-Instruct model via supervised fine-tuning for three epochs on data $(s_t, \phi, a_t^H) \sim \pi^H(\cdot \mid s_t, \phi)$.

Results. Table 2 reports the average success rate of the policy trained with these configurations on the TwentyQuestions test set. We observe that hindsight-guided exploration during critic training contributes to the most significant performance gain $(0.55 \rightarrow 0.77)$. When a critic is ineffective $(\alpha = 0)$, guided exploration during actor training can boost performance $(0.55 \rightarrow 0.62)$. When the critic is trained with a sufficient state-action coverage, additional exploration in actor training does not yield additional gains.

5 Discussion

We present HOPE, a framework that leverages hindsight reasoning in LLMs to guide exploration in multi-turn reinforcement learning. Instead of directly imitating counterfactual actions, HOPE incorporates them into both actor and critic training, enabling iterative policy improvement through a simple and scalable approach. By combining environment rewards with a learned critic, the method identifies beneficial counterfactuals and uses them to shape future behavior.

Experiments show that HOPE consistently outperforms imitation learning and standard RL exploration strategies across domains. Nonetheless, two limitations remain: (1) HOPE inserts a single counterfactual per trajectory before reverting to on-policy rollouts, as full counterfactual rollouts require expensive environment simulation; and (2) evaluations were limited to LLMs with up to three billion parameters, leaving scalability to larger models an open question.

Acknowledgments and Disclosure of Funding

We sincerely thank Esah Shah (es999@cornell.edu) for setting up the GuessMyCity environment and collecting gpt-4o's rollouts. This work was supported in part by the National Science Foundation FRR (#2327973).

References

[1] Marwa Abdulhai, Isadora White, Charlie Snell, Charles Sun, Joey Hong, Yuexiang Zhai, Kelvin Xu, and Sergey Levine. Lmrl gym: Benchmarks for multi-turn reinforcement learning with language models, 2023.

[2] Dilip Arumugam and Thomas L. Griffiths. Toward efficient exploration by large language model agents, 2025.

[3] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2–3):235–256, May 2002.

[4] Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 12461–12495. Curran Associates, Inc., 2024.

[5] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback, 2022.

[6] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

[7] Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 3676–3713. PMLR, 23–29 Jul 2023.

[8] Shicong Cen, Jincheng Mei, Katayoon Goshvadi, Hanjun Dai, Tong Yang, Sherry Yang, Dale Schuurmans, Yuejie Chi, and Bo Dai. Value-incentivized preference optimization: A unified approach to online and offline RLHF. In *The Thirteenth International Conference on Learning Representations*, 2025.

[9] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[10] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning, 2023.

[11] Sanjiban Choudhury and Paloma Sodhi. Better than your teacher: LLM agents that learn from privileged AI feedback. In *The Thirteenth International Conference on Learning Representations*, 2025.

[12] Julian Coda-Forno, Marcel Binz, Zeynep Akata, Matt Botvinick, Jane Wang, and Eric Schulz. Meta-in-context learning in large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 65189–65201. Curran Associates, Inc., 2023.

[13] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[14] Vikranth Dwaracherla, Seyed Mohammad Asghari, Botao Hao, and Benjamin Van Roy. Efficient exploration for LLMs. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 12215–12227. PMLR, 21–27 Jul 2024.

[15] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield,

Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang,

Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024.

[16] Shangmin Guo, Biao Zhang, Tianlin Liu, Tianqi Liu, Misha Khalman, Felipe Llinares, Alexandre Rame, Thomas Mesnard, Yao Zhao, Bilal Piot, Johan Ferret, and Mathieu Blondel. Direct language model alignment from online ai feedback, 2024.

[17] Osband Ian, Van Roy Benjamin, and Russo Daniel. (more) efficient reinforcement learning via posterior sampling. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3003–3011, Red Hook, NY, USA, 2013. Curran Associates Inc.

[18] Arnav Kumar Jain, Gonzalo Gonzalez-Pumariega, Wayne Chen, Alexander M Rush, Wenting Zhao, and Sanjiban Choudhury. Multi-turn code generation through single-step rewards. In *Workshop on Reasoning and Planning for Large Language Models*, 2025.

[19] Geunwoo Kim, Pierre Baldi, and Stephen Marcus McAleer. Language models can solve computer tasks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[20] Akshay Krishnamurthy, Keegan Harris, Dylan J. Foster, Cyril Zhang, and Aleksandrs Slivkins. Can large language models explore in-context?, 2024.

[21] Nicholas Kroeger, Dan Ley, Satyapriya Krishna, Chirag Agarwal, and Himabindu Lakkaraju. Are large language models post hoc explainers? In *R0-FoMo:Robustness of Few-shot and Zero-shot Learning in Large Foundation Models*, 2023.

[22] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning, 2024.

[23] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 21314–21328. Curran Associates, Inc., 2022.

[24] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024.

[25] Zhihan Liu, Hao Hu, Shenao Zhang, Hongyi Guo, Shuqi Ke, Boyi Liu, and Zhaoran Wang. Reason for future, act for now: A principled framework for autonomous llm agents with provable sample efficiency, 2024.

[26] Jianqiao Lu, Zhiyang Dou, Hongru Wang, Zeyu Cao, Jianbo Dai, Yingjia Wan, and Zhijiang Guo. Autopsv: Automated process-supervised verifier, 2024.

[27] Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision, 2024.

[28] Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, and Dong Yu. LASER: LLM agent with state-space exploration for web navigation. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

[29] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594. Curran Associates, Inc., 2023.

[30] Allen Nie, Yi Su, Bo Chang, Jonathan N. Lee, Ed H. Chi, Quoc V. Le, and Minmin Chen. Evolve: Evaluating and optimizing llms for exploration, 2024.

[31] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.

[32] Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents, 2024.

[33] Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025.

[34] Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 55249–55285. Curran Associates, Inc., 2024.

[35] 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, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.

[36] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[37] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[38] Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025.

[39] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc., 2023.

[40] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of LLM agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7584–7600, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[41] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[42] Malcolm J. A. Strens. A bayesian framework for reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, page 943–950, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[43] William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[44] S. Thrun. The role of exploration in learning control. In D.A. White and D.A. Sofge, editors, *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold, Florence, Kentucky 41022, 1992.

[45] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022.

[46] Siddharth Verma, Justin Fu, Sherry Yang, and Sergey Levine. CHAI: A CHatbot AI for task-oriented dialogue with offline reinforcement learning. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz, editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4471–4491, Seattle, United States, July 2022. Association for Computational Linguistics.

[47] Somin Wadhwa, Silvio Amir, and Byron C Wallace. Investigating mysteries of CoT-augmented distillation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 6071–6086, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[48] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[49] Tengyang Xie, Dylan J. Foster, Akshay Krishnamurthy, Corby Rosset, Ahmed Awadallah, and Alexander Rakhlin. Exploratory preference optimization: Harnessing implicit q*-approximation for sample-efficient rlhf, 2024.

[50] Rongwu Xu, Zehan Qi, and Wei Xu. Preemptive answer "attacks" on chain-of-thought reasoning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 14708–14726, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

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

[52] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STar: Bootstrapping reasoning with reasoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[53] Yuexiang Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Shengbang Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, and Sergey Levine. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 110935–110971. Curran Associates, Inc., 2024.

[54] Shenao Zhang, Donghan Yu, Hiteshi Sharma, Han Zhong, Zhihan Liu, Ziyi Yang, Shuohang Wang, Hany Hassan Awadalla, and Zhaoran Wang. Self-exploring language models: Active preference elicitation for online alignment. *Transactions on Machine Learning Research*, 2025.

[55] Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Forty-first International Conference on Machine Learning*, 2024.

Appendix

Table of Contents

A Related Work

A.1 Exploration with LLMs

Early works investigate LLMs' ability to explore under in-context learning, where only the input to the model is updated instead of the model parameters. Some study this ability in multi-armed bandit problems [12, 20, 25, 30], while others investigate whether LLMs can self-refine and explore actions given feedback [5, 19, 29, 39]. However, prompting-based approaches require significant engineering efforts (e.g., by explicitly modeling the structure of the state space [28]).

RLHF [31] provides a practical framework for training and aligning LLMs. To improve the sample efficiency, recent works propose adding an optimistic exploration bonus to the RLHF loss [8, 14, 49, 54], but they are limited to single-turn and are difficult to apply to the multi-turn setting that requires exploration both at the turn level and the token level. Closer to our work are ones that utilize

posterior sampling to guide exploration [2, 14]. Dwaracherla et al. [14] estimate uncertainty via an ensemble of reward models, but it does not directly train the LLM and is also limited to single-turn tasks. Concurrent work [2] uses LLMs to explicitly sample from the posterior over possible MDPs and show positive results for multi-turn deterministic environments. However, they discuss how their approach fails to scale in stochastic domains, which include task-oriented dialogue tasks explored in our work. Instead of explicitly estimating and sampling from the posterior, `HOPE` takes inspiration from works that utilize LLMs' hindsight reasoning ability [21, 47, 50, 52]. Given the summary of a completed trajectory in hindsight, the LLM implicitly estimates a posterior over plausible MDPs and samples possible actions that would be optimal under the MDPs in its reasoning.

A.2 Training LLMs for Multi-Turn Tasks

Reinforcement learning has been leveraged to train LLM agents on reasoning [13, 24, 48], learning self-correction [22, 34], code generation [18, 23], and interactive environments [4, 7, 32, 33, 53]. A class of approaches utilizes rejection sampling to only fine-tune the LLM on successful trajectories [10, 15, 52]. Some works also utilize failed trajectory by learning from expert correction [11] or a contrastive loss [40]. Meanwhile, ARCHER [55] frames the multi-turn problem as a hierarchical MDP where the lower-level MDP considers each token as actions and the higher-level MDP optimizes rewards over turns. In addition to only optimizing sparse reward signals, recent works adopt an actor-critic framework, where they train a critic, or a process reward model, that provides dense supervision [24, 26, 27, 38, 45, 48]. While our work is agnostic to a specific RL algorithm as we focus on improving exploration in multi-turn tasks, we also practically show a simple, scalable implementation of actor-critic RL that trains a critic and an actor over iterations.

B Broader Impacts

Enabling LLMs to learn through online interaction unlocks significant societal benefits. In domains like customer service and software engineering, LLM assistants can automate routine tasks, allowing professionals to focus on higher-order, intellectually demanding problems.

However, self-improving agents introduce risks if not properly constrained. Without well-defined rewards and safeguards, such agents may learn behaviors misaligned with human values even if it is optimizing reward signals. Moreover, these capabilities can be weaponized—malicious actors could make LLM automatically optimize for harmful tasks such as spreading misinformation. Mitigating these risks requires rigorous safety mechanisms, ethical oversight, and robust evaluation protocols to ensure alignment with human interests and societal good.

C Approach Details

Table 3 reports the hyperparameters, the number of gpu/cpu/memory, and estimated training time used during training. We train our models on NVIDIA RTX 6000/NVIDIA RTX 6000 Ada, and we build upon the training code in OpenInstruct[1]. Below, we describe each stage of training: SFT to get π_0 (Section C.1), collecting critic data (Section C.2), iteratively training critic (Section C.3), and iteratively training actor (Section C.4).

C.1 Supervised Finetuning (SFT)

We train Llama-3.2-3B-Instruct on gpt4o rollouts on the training set (and there are 3 trajectories per game). We format the dataset in standard SFT format, where the input includes instruction and the state (the history of observations and actions so far) $s_t = \{o_0, a_0, \ldots, o_t\}$, and the output is the gpt4o's action a_t. Note that we follow ReAct [51] and make a_t contain both the actual action to take and the reasoning for this action.

For CarDealer, because the agent has to first make API calls to the dealership database before talking to the user, the model is trained on equal amount of data with corresponding prompts for both tasks.

[1] `https://github.com/allenai/open-instruct`

Dataset	TwentyQuestion	GuessMyCity	CarDealer
SFT			
batch size	4	4	2
gradient accumulation steps	16	16	12
train epochs		3	
learning rate		3.00e-05	
lr schedular		cosine	
# gpus	2	2	4
# cpus	2	2	4
Mem (GB)	80	80	200
Estimated Time (hrs)	1.25	1.25	1.50
Collecting Trajectory			
# onpolicy traj		14	
policy sample temp		1.0	
# offpolicy traj	16	16	32
# timesteps to sample		2	
sampling range ($T = \texttt{len(traj)}$)	$[0.3T, 0.6T)$	$[0.3T, 0.8T)$	$[2, T-1)$
# counterfactuals at a timestep		2	
hindsight proposer π^H temp		0.3	
Critic Training			
α (% of hindsight data)	0.4	0.5	0.5
batch size		4	
gradient accumulation steps		16	
train epochs		1	
learning rate		5.00e-06	
lr schedular		linear	
# gpus	2	2	4
# cpus	2	2	4
Mem (GB)	80	80	200
Estimated Time (hrs)	2.00	2.00	2.50
Actor Training - OnlineDPO			
batch size		2	
gradient accumulation steps	6	6	1
train epochs		1	
learning rate		8.00e-08	
lr schedular		linear	
generation temp		0.7	
# gpus (1 used for generator)	4	4	6
# cpus	2	2	4
Mem (GB)	200	200	250
Estimated Time (hrs)	2.25	3.00	3.40
Shared parameters			
max seq length	2048	3000	3500
optimizer		AdamW	

Table 3: Training hyperparameters and estimated training time.

C.2 Collecting Trajectories

We leverage a fast inference library, SG-Lang[2] to serve both the environment simulator and the current policy efficiently.

For all domains, we first collect 14 trajectories per game on the training set via the current policy with a temperature of 1.0. Then, we randomly sample 4 trajectories to augment. For each of these trajectories, we randomly sample 2 timesteps based on a domain-specific range with respect to the trajectory length. On a first principle, for domains that require search (TwentyQuestions, GuessMyCity), the counterfactual action is not useful when the timestep is too early (when the policy can already eliminate common categories) or when the timestep is too late (when the proposer can directly end the game by guess the word). At each sampled timestep, we use the hindsight proposer to generate 2 distinct counterfactual actions. Then, we augment the original trajectory by splicing in the counterfactuals before completing the trajectory with the original policy. Thus, with all the parameters, we generate $4 \times 2 \times 2 = 16$ hindsight trajectories.

Note that, because CarDealer requires generating two-part counterfactuals (first 2 counterfactual API calls, then 2 counterfactual responses to the user), we generate $4 \times 2 \times (2 + 2) = 32$ hindsight trajectories in total.

C.3 Training Critic via Supervised Learning

We fix the dataset to 10k datapoints for TwentyQuestions/GuessMyCity and 20k datapoints for CarDealer (because we have 10k datapoints for making API call and 10k datapoints for responding to buyer).

After calculating the Q-values via MC estimate, we normalize the Q-targets to be between $[0, 1]$. To maintain a balance dataset, we ensure that 50% of datapoints have low values $[0, 0.5)$ and 50% of datapoints have high values $[0.5, 1]$.

- For low-value data, we prioritize using datapoints from on-policy trajectories (i.e., trajectories that are generated by only sampling from the current policy). If there isn't enough datapoints, we add hindsight trajectories until sufficient.
- For high-value data, we use α to control the percentage of datapoints from hindsight trajectories.

Critic is initialized with weights from π_0. It has an additional randomly initialized linear layer of dimension $[\texttt{hidden\_dim}, 1]$ because it predicts a scalar value. To select the best intermediate checkpoint for actor training, we evaluate the Best-of-N performance of using the current policy π_{i-1} as the generator and the checkpoint as the critic. Specifically, at each step, π_{i-1} generates ($N = 15$) actions at temperature 0.7, and we execute the action with the highest score from the critic. The best critic is one that has the highest Best-of-N success rate on the validation set.

C.4 Training Actor via OnlineDPO

Similar to the critic, the actor is initialized with weights from π_0. The parameter β controls the probability of generating counterfactual actions from the hindsight proposer. For most experiments, we set $\beta = 0.0$. When we use hindsight-guided action experation, we set $\beta = 0.5$

Due to budget constraints, we distill the GPT-4o hindsight proposer into a Llama-3.2-3B-Instruct. To create the training data, we first consolidate all the counterfactual actions generated during Section C.2 and use SFT to fully fine-tune Llama-3.2-3B-Instruct for 2 epoches using learning rate 3e-5. Then, we further fine-tune the model on only counterfactuals that lead to success for another epoch using learning 3e-6. The other training hyperparameters are the same as the one for SFT in Table 3.

C.4.1 Evaluation

For all domains, we evaluate the policy once per game on the training set. For TwentyQuestions and CarDealer, we evaluate the policy three times per game on the validation and test set. For GuessMyCity, we evaluate the policy ten times per game on the validation and test set. We always select the intermediate checkpoint that has the average highest validation success rate.

[2] https://github.com/sgl-project/sglang

						LEAP		Reject-S		PRM+RL		HOPE	
			gpt4o	3B	π_0	π_1	π_2	π_1	π_2	π_1	π_2	π_1	π_2
TwentyQuestions	Train	Return	1.00 (0.09)	0.06 (0.02)	1.04 (0.09)	1.13 (0.09)	1.15 (0.09)	1.25 (0.08)	1.24 (0.09)	1.53 (0.08)	1.34 (0.09)	1.28 (0.09)	1.29 (0.09)
		Success	0.60 (0.04)	0.16 (0.03)	0.66 (0.04)	0.75 (0.04)	0.77 (0.04)	0.79 (0.04)	0.79 (0.04)	0.86 (0.03)	0.81 (0.04)	0.77 (0.04)	0.80 (0.04)
	Val	Return	1.00 (0.09)	0.04 (0.02)	1.01 (0.05)	1.21 (0.09)	1.00 (0.11)	1.22 (0.10)	1.07 (0.09)	1.38 (0.05)	1.10 (0.05)	1.18 (0.04)	1.34 (0.05)
		Success	0.65 (0.05)	0.16 (0.04)	0.62 (0.02)	0.83 (0.04)	0.64 (0.05)	0.79 (0.04)	0.79 (0.04)	0.79 (0.02)	0.79 (0.02)	0.77 (0.02)	0.78 (0.02)
	Test	Return	1.00 (0.11)	0.09 (0.04)	1.08 (0.11)	1.16 (0.12)	0.52 (0.09)	0.63 (0.11)	0.90 (0.13)	1.19 (0.16)	1.05 (0.09)	1.15 (0.13)	1.91 (0.09)
		Success	0.61 (0.06)	0.28 (0.05)	0.67 (0.05)	0.68 (0.06)	0.45 (0.06)	0.55 (0.06)	0.63 (0.06)	0.55 (0.06)	0.82 (0.05)	0.77 (0.05)	0.97 (0.02)
	Total	Return	1.00 (0.06)	0.06 (0.01)	1.03 (0.04)	1.16 (0.06)	0.95 (0.06)	1.09 (0.06)	1.10 (0.06)	**1.39 (0.04)**	1.14 (0.04)	1.19 (0.04)	**1.39 (0.04)**
		Success	0.62 (0.03)	0.19 (0.02)	0.63 (0.02)	0.76 (0.03)	0.65 (0.03)	0.73 (0.03)	0.75 (0.03)	0.78 (0.02)	0.80 (0.02)	0.77 (0.02)	**0.80 (0.02)**
GuessMyCity	Train	Return	1.00 (0.11)	0.03 (0.01)	0.93 (0.09)	0.60 (0.06)	0.39 (0.04)	0.50 (0.04)	0.52 (0.04)	0.36 (0.03)	0.63 (0.07)	0.37 (0.03)	0.87 (0.09)
		Success	0.58 (0.04)	0.04 (0.02)	0.72 (0.04)	0.70 (0.04)	0.66 (0.04)	0.76 (0.04)	0.81 (0.04)	0.66 (0.04)	0.65 (0.04)	0.64 (0.04)	0.71 (0.04)
	Val	Return	1.00 (0.11)	0.07 (0.03)	0.74 (0.05)	0.40 (0.05)	0.26 (0.03)	0.41 (0.02)	0.35 (0.02)	0.27 (0.01)	0.42 (0.02)	0.32 (0.03)	0.69 (0.07)
		Success	0.66 (0.05)	0.10 (0.03)	0.74 (0.03)	0.68 (0.05)	0.59 (0.05)	0.79 (0.02)	0.78 (0.02)	0.72 (0.02)	0.73 (0.02)	0.74 (0.05)	0.78 (0.04)
	Test	Return	1.00 (0.12)	0.01 (0.01)	0.80 (0.09)	0.35 (0.04)	0.29 (0.03)	0.46 (0.02)	0.31 (0.03)	0.35 (0.05)	0.54 (0.03)	0.45 (0.05)	0.91 (0.10)
		Success	0.56 (0.05)	0.01 (0.01)	0.70 (0.05)	0.59 (0.05)	0.62 (0.05)	0.69 (0.02)	0.62 (0.05)	0.66 (0.05)	0.66 (0.02)	0.74 (0.05)	0.77 (0.04)
	Total	Return	**1.00 (0.07)**	0.03 (0.01)	0.80 (0.04)	0.47 (0.03)	0.32 (0.02)	0.44 (0.01)	0.37 (0.01)	0.29 (0.01)	0.50 (0.02)	0.38 (0.02)	0.83 (0.05)
		Success	0.59 (0.03)	0.05 (0.01)	0.73 (0.02)	0.66 (0.03)	0.63 (0.03)	0.74 (0.01)	**0.77 (0.02)**	0.70 (0.02)	0.69 (0.02)	0.70 (0.03)	0.75 (0.03)
CarDealer (Tool)	Train	Return	1.00 (0.10)	0.33 (0.07)	1.27 (0.11)	1.17 (0.11)	1.27 (0.10)	1.23 (0.11)	1.27 (0.10)	1.31 (0.11)	1.31 (0.11)	1.38 (0.11)	1.34 (0.10)
		Success	0.46 (0.04)	0.30 (0.04)	0.54 (0.04)	0.50 (0.05)	0.57 (0.04)	0.54 (0.04)	0.55 (0.04)	0.54 (0.04)	0.54 (0.04)	0.58 (0.04)	0.55 (0.04)
	Val	Return	1.00 (0.12)	0.68 (0.11)	1.23 (0.13)	1.35 (0.13)	1.44 (0.12)	1.40 (0.13)	1.52 (0.12)	1.51 (0.13)	1.57 (0.13)	1.61 (0.13)	1.62 (0.13)
		Success	0.40 (0.05)	0.34 (0.05)	0.47 (0.05)	0.54 (0.05)	0.62 (0.05)	0.56 (0.05)	0.62 (0.05)	0.58 (0.05)	0.60 (0.05)	0.64 (0.05)	0.62 (0.05)
	Test	Return	1.00 (0.09)	0.40 (0.07)	1.03 (0.08)	0.77 (0.08)	0.86 (0.08)	1.07 (0.08)	1.00 (0.08)	1.00 (0.08)	1.14 (0.08)	1.07 (0.08)	1.20 (0.08)
		Success	0.59 (0.05)	0.51 (0.05)	0.66 (0.05)	0.52 (0.05)	0.57 (0.05)	0.68 (0.05)	0.62 (0.05)	0.67 (0.05)	0.72 (0.04)	0.69 (0.05)	0.77 (0.04)
	Total	Return	1.00 (0.06)	0.46 (0.05)	1.18 (0.06)	1.10 (0.07)	1.19 (0.06)	1.24 (0.06)	1.27 (0.06)	1.28 (0.06)	1.34 (0.06)	1.36 (0.06)	**1.38 (0.06)**
		Success	0.48 (0.03)	0.38 (0.03)	0.55 (0.03)	0.52 (0.03)	0.59 (0.03)	0.59 (0.03)	0.60 (0.03)	0.60 (0.03)	0.62 (0.03)	0.63 (0.03)	**0.64 (0.03)**

Table 4: **Policy performance on all data splits of three task-oriented conversational environments.** We report the average normalized return and success rate with standard error, and we highlight the top-performing approaches. The return is normalized with respect to gpt4o's performance.

D Additional Experimental Results

D.1 What is the optimal ratio between on-policy and hindsight data in the replay buffer?

We study how the proportion of hindsight data influences critic training by varying the data mixture used in Algorithm 2. For each setting, we construct datasets with a fixed size of 10k transitions but vary the ratio of on-policy to hindsight-generated data. As shown in Figure 3, incorporating even a small amount of hindsight data substantially improves performance: adding just 20–40% hindsight data increases π_1's success rate from 0.55 (the PRM+RL baseline with no hindsight) to over 0.70. Performance quickly plateaus, and pure hindsight data does not yield additional gains. In practice, we use a 60% hindsight and 40% on-policy mixture for the Twenty Questions domain, which achieves the highest observed success rate of 0.77. Note that this setting is not purely off-policy—the critic still sees some on-policy data from π_0, which stabilizes learning and anchors value estimates around feasible behaviors.

D.2 Full Results

Table 4 shows the complete result of all approaches performance on different data split. For TwentyQuestions and CarDealer, HOPE outperform all baselines including gpt4o, achieving the highest normalized return and success rate in total. For GuessMyCity, although HOPE has slightly lower normalized returns (0.83) compared to gpt4o (1.00), it still outperforms the remaining baselines. Similarly, it has the second highest success rate (0.75 ± 0.03) that is within standard error of the highest success rate (0.77 ± 0.02).

Overall, other approaches tend to overfit on the train set, while HOPE is able to maintain high performance on the test set. For example, for TwentyQuestions, although PRM+RL π_1 achieves the highest normalized return (1.53) and success rate (1.38) on the training set, it significantly underperforms on the test set with normalized return of 1.19 and a lower success rate 0.55 than π_0 (0.67). In contrast, although HOPE has

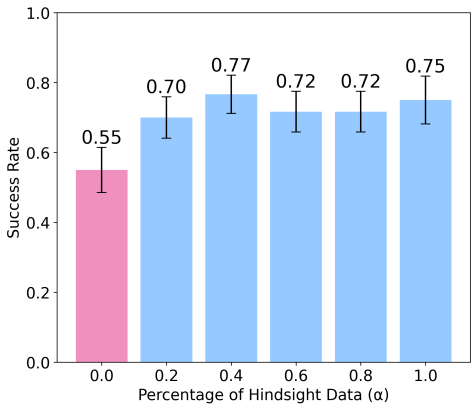


Figure 3: **Effect of critic data mixture on downstream policy performance**. We vary the percentage of hindsight data via the hyperparameter α and train the corresponding critic and $\pi 1$. We report the average success with standard error in TwentyQuestions.

the third highest normalized return (1.29) and success rate (0.80), it maintains the highest normalized return (1.91) and success rate (0.97), significantly outperforming all other baselines.

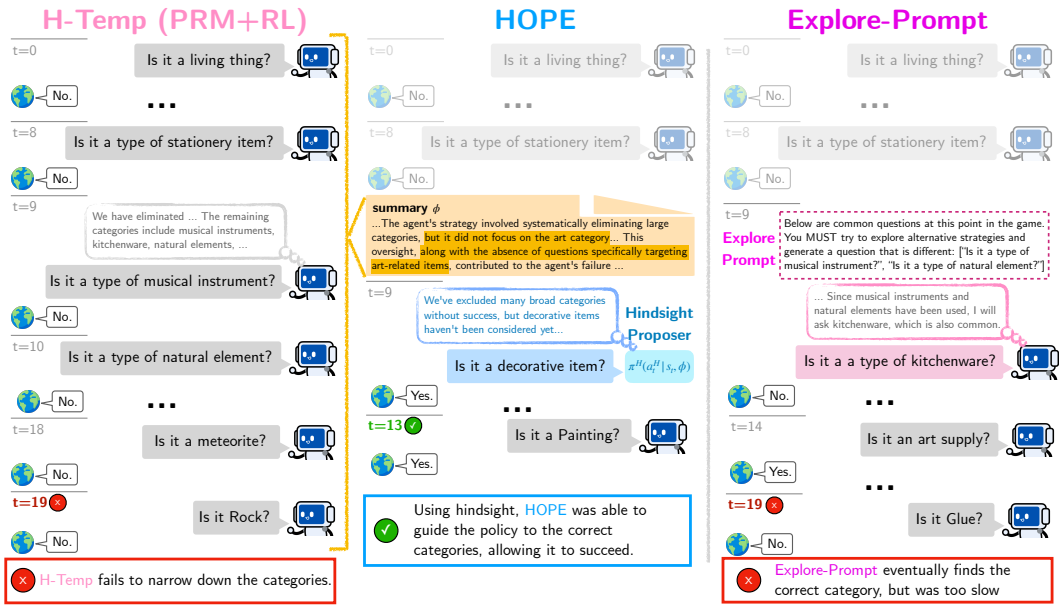


Figure 4: **Qualitative Example of how HOPE improves exploration during critic data collection.** The three columns shows traces of different exploration strategies collecting data for the same game (where the secret word is "Painting". In the 1st column, H-Temp (PRM+RL) simply sample actions from the policy with high temperature. In the 2nd column, HOPE first summarize the completed rollout before using the hindsight proposer to generate counterfactual actions. In the 3rd column, Explore-Prompt explicitly prompt the policy to generate actions different from the common one.

D.3 Qualitative: How does hindsight help exploration?

Figure 4 demonstrations how HOPE effectively explore the state and action space, leading to successful trajectories, compared to other exploration strategies.

In the 1st column, H-Temp (PRM+RL) simply samples the current policy (in this case π_0) at a high temperature of 1.0. The policy is ineffective at exploring the high-value state space and action space. Although it attempts to eliminate high level categories, it fails to consider categories such as "art supply", leading the policy to exhaust the maximum number of questions that it can ask.

In the 3rd column, Explore-Prompt first samples 5 actions from the current policy. Then, given this set of high-probability actions, prompt asks the policy to generate some alternative actions that differ from the list. The policy is able to identify the correct category, but the policy still takes too long and uses all the questions before being able to guess "Painting." Although Explore-Prompt can increase the diversity of states and actions visited, it fails to intentionally explore high-value regions of the state and action space.

In contrast, in the 2nd column, HOPE first accurately identifies the original policy's mistake in its summary ϕ of the completed trajectory: "it did not focus on the art category... this oversight ... contributes to the agent's failure." Then, the hindsight proposer generates an effective counterfactual action "is it a decorative item?", which leads to the policy quickly guess the correct word at timestep $t = 13$.

E Domain: TwentyQuestions Details

E.1 Environment Setup

Training set contains 15 object categories with 110 objects in total. Validation set has unseen objects in the train categories with 28 objects in total. Test set has 2 unseen object categories with 20 objects.

When the agent successfully guess the word, the reward is 0. Otherwise, the reward is -1 to encourage the agent to complete the game as soon as possible. The agent has maximum of 20 steps to complete the game.

E.2 Agent Prompt

When there is only one question left, we programmatically change the mode to 'input_final', which require the agent to guess a specific object.

```
{% if mode == 'input' %}
You are an intelligent player playing a game of twenty questions. Your
↪ objective is to ask the minimal number of yes-no questions in order to
↪ guess the identity of the entity/object chosen by an oracle. You can
↪ only ask 20 questions in total.

The entity/object that the orcacle can choose from are:
{{ all_obj_list }}

Your goal is to generate a yes-no question that either (1) help you narrow
↪ down the possible things as much as possible or (2) guess the entity/
↪ object directly.

Please follow these general instructions:
* You MUST ask a yes-no question.
* If you are guessing the entity/object directly, you MUST ask a question
↪ in the format "Is it {your_guess}?". your_guess must be one of the
↪ entity/object that the oracle can choose from.
* Before you ask the question, you MUST intelligently reason about the
↪ history of previous questions and answers to ask the most informative
↪ question. Your reasoning should also be based on the list of entity/
↪ object that the oracle can choose from.
* Consult the history of previous questions and answers to see what
↪ questions you have asked already so as to not repeat your questions.
* Do NOT repeat the same question. It's going to yield the same result.
* Do NOT get stuck on one idea and try to branch out if you get stuck.

Below is the history of previous questions and answers:
{{ observation_action_history }}

You MUST generate a response in the following format. Please issue only a
↪ single question at a time.
REASON:
Rationale for what question to ask next based on the previous history and
↪ the list of things that the oracle can choose from.
QUESTION:
The question to be asked. The question must be a yes-no question.
{% elif mode == 'input_final' %}
You are an intelligent player playing a game of twenty questions. Your
↪ objective is to ask the minimal number of yes-no questions in order to
↪ guess the identity of the entity/object chosen by an oracle. You can
↪ only ask 20 questions in total.

The entity/object that the orcacle can choose from are:
```

```
{{ all_obj_list }}

Your goal is to generate a yes-no question that either (1) help you narrow
↪ down the possible things as much as possible or (2) guess the entity/
↪ object directly.

Please follow these general instructions:
* You MUST ask a yes-no question.
* If you are guessing the entity/object directly, you MUST ask a question
↪ in the format "Is it {your_guess}?". your_guess must be one of the
↪ entity/object that the oracle can choose from.
* Before you ask the question, you MUST intelligently reason about the
↪ history of previous questions and answers to ask the most informative
↪ question. Your reasoning should also be based on the list of entity/
↪ object that the oracle can choose from.
* Consult the history of previous questions and answers to see what
↪ questions you have asked already so as to not repeat your questions.
* Do NOT repeat the same question. It's going to yield the same result.
* Do NOT get stuck on one idea and try to branch out if you get stuck.

Below is the history of previous questions and answers:
{{ observation_action_history }}

You have already asked 19 questions, so this is your final guess. Your goal
↪  is to consult the list of entity/object that the orcale can choose from
↪  and quess the entity/object directly.

You MUST generate a response in the following format. Please issue only a
↪ single question at a time.
REASON:
Rationale for what entity/object to quess based on the previous history. In
↪  your reason, consult the list of entity/object that the oracle can
↪ choose from to precisely state VERBATIM what you will guess.
QUESTION:
The question to be asked. The question must be a yes-no question in the
↪ format "Is it {your_guess}?". your_guess must be one of the entity/
↪ object that the oracle can choose from.
{% elif mode == 'output' %}
REASON:
{{ reason }}
QUESTION:
{{ action }}
{% elif mode == 'output_no_reason' %}
QUESTION:
{{ action }}
{% endif %}
```

E.3 Summary Prompt

```
{% if system %}
You are an intelligent assistant summarizing a game of twenty questions,
↪ where an agent is trying to guess a word by asking at most 20 yes-no
↪ questions.

## Overall information about the game
Here is the list of possible secret words:
{{ all_obj_list }}
```

```
## What you receive as input
You are given (1) a chat history of the questions and answers and (2) the
↪ secret word that the agent is trying to guess.

## Your goal and rules to follow
You must summarize the chat history in 2-4 sentences (what the agent asked,
↪  whether the agent succeeded or not). You must also mention what the
↪ actual secret word is and what general category the secret word is in.

You should also discuss in your summary the strategy of the agent. You can
↪ refer to the list of possible secret words to help with your discussion:
* Did the agent repeat similar questions that seem less helpful?
* Did the agent move on to ask about specific things/entities too quickly?
↪ Or did the agent keep asking broader, higher-level questions even though
↪  they can directly guess the word with the information it had?
* If the agent succeeded, what type of questions did it ask to help it
↪ succeed?
* If the agent failed, what are some possible reasons on why it failed? Did
↪  it ask questions that violate previous questions and answers? Did it
↪ ask about things not in the list of possible secret words?

You can directly generate the summary as a paragraph. You should not add
↪ any formatting (e.g., markdown) when generating the summary.
{% endif %}
{% if not system %}
{% if mode == 'input' %}
## Chat History
{{ observation_action_history }}

## Secret Word
{{ goal }}
{% endif %}
{% endif %}
```

E.4 Hindsight Proposer Prompt

```
{% if system %}
You are an intelligent teacher who gives guidance on what question to ask
↪ in a game of twenty questions. In a game of twenty questions, a player
↪ is trying to guess a secret word by asking at most 20 yes-no questions.

## What you receive as input
You are given (1) a list of the possible secret words and (2) the chat
↪ history of the questions that the player has asked so far and the
↪ corresponding answers. To further help you make wise judgements and
↪ provide helpful guidance to the player, you are also given (3) a summary
↪  of the complete game (which includes whether the player has succeeded
↪ in the end, the actual secret word, and the general category that the
↪ secred word is in).

## Your goal and rules to follow
Your goal is to use your hindsight reasoning ability to generate {{
↪ num_responses}} alternative questions that the player should have asked
↪ given the current chat history. Your process is to: 1. reason about all
↪ the information (including the summary); 2. generate some questions that
↪  the player could have asked; 3. generate some plausible reasoning that
↪ the player could have come up with based on the current chat history.
```

```
Please follow these general instructions:
- **Ask feasible questions:** The question that you ask MUST be a question
↪ that is possible for the player to ask given the current chat history.
- **In teacher_reason, reason with all the information:** You are the
↪ teacher. In your teacher reasoning, you MUST make reference to specific
↪ things mentioned under ### List of the possible secret words and the
↪ chat history. You MUST ask a question that is possible and reasonable to
↪  ask given the current chat history. You can make use of the ### Summary
↪  of the entire game to help you identify better but STILL FEASIBLE
↪ question to ask given the current chat history. For example, the general
↪  category that the secret word is in could help inform you what kind of
↪ question to ask.
- **In player_reason, reasoning should not include secret information from
↪ the summary:"** You are generating what is the possible reasoning that a
↪  player can have based on the chat history in order to generate the
↪ question. The reasoning must not reveal that you know the secret object
↪ or the general category. It must be a feasible reasoning based on the
↪ chat history alone.
- **Question should only be asking about the possible secret words:** Your
↪ question MUST only ask about objects in ### List of the possible secret
↪ words.
- **Ask a new question.** You MUST NOT simply repeat previously asked
↪ questions. You MUST NOT simply combine multiple previously asked
↪ questions.
- If you think it is reasonable and feasible to directly guess the object
↪ given the current chat history, you can propose the question to directly
↪  ask: "Is it {your_guess}?". your_guess must be one of the words in ###
↪ List of the possible secret words.

## Output format
The output is a list of JSON containing {{num_responses}} different pairs
↪ of reasoning and feasible question that you would have asked.
‘‘‘json
[
    {
        "teacher_reason": "string: your rationale for the question that you
        ↪ think the player should have asked instead.",
        "question": "string: your question",
        "player_reason": "string: If you are the player who does not know
        ↪ the secret object or the general category that the object is in,
        ↪ what will be your reasoning in order to generate the action? You
        ↪ MUST only refer to the chat history. You MUST NOT talk about the
        ↪ summary, which the player does not have access to. You MUST NOT
        ↪ reveal what the secret object is. You MUST NOT reveal what the
        ↪ general category that the secret object is in.",
    }
    ...
]
‘‘‘

## Overall information about the current game
### List of the possible secret words
{{ all_obj_list }}

### Summary of the entire game
{{ summary }}
{% endif %}
{% if not system %}
{% if mode == 'input' %}
```

```
Here is the chat history:
{{ observation_action_history }}

At this point in the chat history, what question would you have asked? You
↪ MUST generate {{num_responses}} different, diverse questions.
{% endif %}
{% endif %}
```

F Domain: GuessMyCity Details

F.1 Environment Setup

Training set contains 15 countries, each with 8 cities, so it has 120 cities in total. Validation set has 2 more unseen cities in the 15 training countries, so it has 30 cities in total. Test set has 3 unseen countries, each with 10 cities, so it has 30 cities.

Because the agent can now ask open-ended questions, it is able to extract much more information per turn compared to Twenty Questions. Thus, we limit the maximum step to only 10 steps. In addition, we only allow the agent to only guess the city once. If the agent guesses incorrectly, the episode immediately terminates, and the agent receives a penalty such that its sum of reward is -10. If the agent succeed, it receives a reward of 0. All non-terminating steps get a reward of -1.

F.2 Agent Prompt

```
{% if mode == 'input' %}
You are an intelligent player playing a game where you must guess where
↪ someone is from. Your objective is to ask the minimal number of open-
↪ ended questions that are diverse in order to accurately guess their city
↪  of origin. You can ask 10 questions to determine where they are from.
↪ You are smart, so you will ask the question that will narrow down the
↪ possible cities the person is from as much as possible.

The cities that the oracle can choose from are:
{{ all_city_list }}

Your goal is to generate an open-ended question that (1) helps you narrow
↪ down the possible cities as much as possible or (2) allows you to make a
↪  final guess.

Please follow these general instructions:
* You MUST ask an open-ended question.
* You CANNOT ask the oracle for the name of the city or country.
* If you are guessing the city directly, you MUST ask a question in the
↪ format "Is the city {your_guess}?". your_guess must be one of the cities
↪  that the oracle can choose from.
* Before you ask the question, you MUST intelligently reason about the
↪ history of previous questions and answers to ask the most informative
↪ question.
* Consult the history of previous questions and answers to avoid repeating
↪ the same question.
* Do NOT repeat the same question. It's going to yield the same result.
* Do NOT get stuck on one idea and try to branch out if you get stuck.
* Do NOT ask questions formatted like "Can you ..."
* If you are certain you know the answer, MAKE A GUESS

Below is the history of previous questions and answers:
{{ observation_action_history }}
```

```
You MUST generate a response in the following format. Please issue only a
↪ single question at a time.

REASON:
Rationale for what question to ask next based on the previous history and
↪ the list of cities the oracle can choose from.

QUESTION:
The question to be asked. The question must be open ended.

{% elif mode == 'input_final' %}
You are an intelligent player playing a game where you must guess where
↪ someone is from. Your objective is to ask the minimal number of open-
↪ ended questions that are diverse in order to accurately guess their city
↪  of origin. You can ask 10 questions to determine where they are from.
↪ You are smart, so you will ask the question that will narrow down the
↪ possible cities the person is from as much as possible.

The cities that the oracle can choose from are:
{{ all_city_list }}

Your goal is to generate an open-ended question that (1) helps you narrow
↪ down the possible cities as much as possible or (2) allows you to make a
↪  final guess.

Please follow these general instructions:
* You MUST ask an open-ended question.
* You CANNOT ask the oracle for the name of the city or country.
* If you are guessing the city directly, you MUST ask a question in the
↪ format "Is it {your_guess}?". your_guess must be one of the cities that
↪ the oracle can choose from.
* Before you ask the question, you MUST intelligently reason about the
↪ history of previous questions and answers to ask the most informative
↪ question.
* Consult the history of previous questions and answers to avoid repeating
↪ the same question.
* Do NOT repeat the same question. It's going to yield the same result.
* Do NOT get stuck on one idea and try to branch out if you get stuck.

Below is the history of previous questions and answers:
{{ observation_action_history }}

You have already asked 9 questions, so this is your final guess. Your goal
↪ is to consult the list of cities that the orcale can choose from and
↪ guess the city directly.

You MUST generate a response in the following format. Please issue only a
↪ single question at a time.

REASON:
Rationale for what city to guess based on the previous history. In your
↪ reason, consult the list of cities that the oracle can choose from to
↪ precisely state VERBATIM what you will guess.

QUESTION:
The question to be asked. The question must be a yes-no question in the
↪ format "Is the city {your_guess}?". your_guess must be one of the cities
↪  that the oracle can choose from and be based on what you know about the
↪  city so far.
```

```
{% elif mode == 'output' %}
REASON:
{{ reason }}
QUESTION:
{{ action }}

{% elif mode == 'output_no_reason' %}
QUESTION:
{{ action }}
{% endif %}
```

F.3 Summary Prompt

```
{% if mode == 'input' %}
You are an intelligent player playing a game where you must guess where
↪ someone is from. Your objective is to ask the minimal number of open-
↪ ended questions that are diverse in order to accurately guess their city
↪  of origin. You can ask 10 questions to determine where they are from.
↪ You are smart, so you will ask the question that will narrow down the
↪ possible cities the person is from as much as possible.

The cities that the oracle can choose from are:
{{ all_city_list }}

Your goal is to generate an open-ended question that (1) helps you narrow
↪ down the possible cities as much as possible or (2) allows you to make a
↪  final guess.

Please follow these general instructions:
* You MUST ask an open-ended question.
* You CANNOT ask the oracle for the name of the city or country.
* If you are guessing the city directly, you MUST ask a question in the
↪ format "Is the city {your_guess}?". your_guess must be one of the cities
↪  that the oracle can choose from.
* Before you ask the question, you MUST intelligently reason about the
↪ history of previous questions and answers to ask the most informative
↪ question.
* Consult the history of previous questions and answers to avoid repeating
↪ the same question.
* Do NOT repeat the same question. It's going to yield the same result.
* Do NOT get stuck on one idea and try to branch out if you get stuck.
* Do NOT ask questions formatted like "Can you ..."
* If you are certain you know the answer, MAKE A GUESS

Below is the history of previous questions and answers:
{{ observation_action_history }}

You MUST generate a response in the following format. Please issue only a
↪ single question at a time.

REASON:
Rationale for what question to ask next based on the previous history and
↪ the list of cities the oracle can choose from.

QUESTION:
The question to be asked. The question must be open ended.
```

```
{% elif mode == 'input_final' %}
You are an intelligent player playing a game where you must guess where
↪ someone is from. Your objective is to ask the minimal number of open-
↪ ended questions that are diverse in order to accurately guess their city
↪  of origin. You can ask 10 questions to determine where they are from.
↪ You are smart, so you will ask the question that will narrow down the
↪ possible cities the person is from as much as possible.

The cities that the oracle can choose from are:
{{ all_city_list }}

Your goal is to generate an open-ended question that (1) helps you narrow
↪ down the possible cities as much as possible or (2) allows you to make a
↪  final guess.

Please follow these general instructions:
* You MUST ask an open-ended question.
* You CANNOT ask the oracle for the name of the city or country.
* If you are guessing the city directly, you MUST ask a question in the
↪ format "Is it {your_guess}?". your_guess must be one of the cities that
↪ the oracle can choose from.
* Before you ask the question, you MUST intelligently reason about the
↪ history of previous questions and answers to ask the most informative
↪ question.
* Consult the history of previous questions and answers to avoid repeating
↪ the same question.
* Do NOT repeat the same question. It's going to yield the same result.
* Do NOT get stuck on one idea and try to branch out if you get stuck.

Below is the history of previous questions and answers:
{{ observation_action_history }}

You have already asked 9 questions, so this is your final guess. Your goal
↪ is to consult the list of cities that the orcale can choose from and
↪ guess the city directly.

You MUST generate a response in the following format. Please issue only a
↪ single question at a time.

REASON:
Rationale for what city to guess based on the previous history. In your
↪ reason, consult the list of cities that the oracle can choose from to
↪ precisely state VERBATIM what you will guess.

QUESTION:
The question to be asked. The question must be a yes-no question in the
↪ format "Is the city {your_guess}?". your_guess must be one of the cities
↪  that the oracle can choose from and be based on what you know about the
↪  city so far.

{% elif mode == 'output' %}
REASON:
{{ reason }}
QUESTION:
{{ action }}

{% elif mode == 'output_no_reason' %}
QUESTION:
{{ action }}
```

```
{% endif %}
```

F.4 Hindsight Proposer Prompt

```
{% if mode == 'input' %}
You are an intelligent player playing a game where you must guess where
↪ someone is from. Your objective is to ask the minimal number of open-
↪ ended questions that are diverse in order to accurately guess their city
↪  of origin. You can ask 10 questions to determine where they are from.
↪ You are smart, so you will ask the question that will narrow down the
↪ possible cities the person is from as much as possible.

The cities that the oracle can choose from are:
{{ all_city_list }}

Your goal is to generate an open-ended question that (1) helps you narrow
↪ down the possible cities as much as possible or (2) allows you to make a
↪  final guess.

Please follow these general instructions:
* You MUST ask an open-ended question.
* You CANNOT ask the oracle for the name of the city or country.
* If you are guessing the city directly, you MUST ask a question in the
↪ format "Is the city {your_guess}?". your_guess must be one of the cities
↪  that the oracle can choose from.
* Before you ask the question, you MUST intelligently reason about the
↪ history of previous questions and answers to ask the most informative
↪ question.
* Consult the history of previous questions and answers to avoid repeating
↪ the same question.
* Do NOT repeat the same question. It's going to yield the same result.
* Do NOT get stuck on one idea and try to branch out if you get stuck.
* Do NOT ask questions formatted like "Can you ..."
* If you are certain you know the answer, MAKE A GUESS

Below is the history of previous questions and answers:
{{ observation_action_history }}

You MUST generate a response in the following format. Please issue only a
↪ single question at a time.

REASON:
Rationale for what question to ask next based on the previous history and
↪ the list of cities the oracle can choose from.

QUESTION:
The question to be asked. The question must be open ended.

{% elif mode == 'input_final' %}
You are an intelligent player playing a game where you must guess where
↪ someone is from. Your objective is to ask the minimal number of open-
↪ ended questions that are diverse in order to accurately guess their city
↪  of origin. You can ask 10 questions to determine where they are from.
↪ You are smart, so you will ask the question that will narrow down the
↪ possible cities the person is from as much as possible.

The cities that the oracle can choose from are:
{{ all_city_list }}
```

```
Your goal is to generate an open-ended question that (1) helps you narrow
↪ down the possible cities as much as possible or (2) allows you to make a
↪  final guess.

Please follow these general instructions:
* You MUST ask an open-ended question.
* You CANNOT ask the oracle for the name of the city or country.
* If you are guessing the city directly, you MUST ask a question in the
↪ format "Is it {your_guess}?". your_guess must be one of the cities that
↪ the oracle can choose from.
* Before you ask the question, you MUST intelligently reason about the
↪ history of previous questions and answers to ask the most informative
↪ question.
* Consult the history of previous questions and answers to avoid repeating
↪ the same question.
* Do NOT repeat the same question. It's going to yield the same result.
* Do NOT get stuck on one idea and try to branch out if you get stuck.

Below is the history of previous questions and answers:
{{ observation_action_history }}

You have already asked 9 questions, so this is your final guess. Your goal
↪ is to consult the list of cities that the orcale can choose from and
↪ guess the city directly.

You MUST generate a response in the following format. Please issue only a
↪ single question at a time.

REASON:
Rationale for what city to guess based on the previous history. In your
↪ reason, consult the list of cities that the oracle can choose from to
↪ precisely state VERBATIM what you will guess.

QUESTION:
The question to be asked. The question must be a yes-no question in the
↪ format "Is the city {your_guess}?". your_guess must be one of the cities
↪  that the oracle can choose from and be based on what you know about the
↪  city so far.

{% elif mode == 'output' %}
REASON:
{{ reason }}
QUESTION:
{{ action }}

{% elif mode == 'output_no_reason' %}
QUESTION:
{{ action }}
{% endif %}
```

G Domain: CarDealer Details

G.1 Environment Setup

We improve the original CarDealer environment in LMRL [1] to include tool use and simulated users whose preferences/constraints evolve in the interaction. There are 10 steps to complete the task. At

each step, the agent must perform a two-part action: making API calls and generating response to the user.

Action (Part 1): API calls. At each step, the agent must choose one of the following API calls:

- `search_car_by_brand_type(car_brand:str, car_type:str)`
- `search_car_by_brand(car_brand:str)`
- `search_car_by_type(car_type:str)`
- `search_car_that_have_features(features_list:List[str])`. Note that this finds the cars that have at least all the features specified in the `features_list`. These cars could have additional features.
- `no_op()`

To limit the input length to the agent, we only show a maximum of 8 cars. Each shown car has information about its brand, type, features, market price (MSRP), and suggested discounts that the agent can use.

Action (Part 2): Reply to Buyer. After the API calls, the agent has access to a list of cars from the database. The agent must generate a reply to the buyer and select a car from the list of cars if it wants to propose a car to the user. Because we empirically observe that open models, such as Llama-3.2-3B-Instruct, has a tendency to hallucinate cars not in the database, we also require the agent to copy down all the information about the car that it is proposing. When negotiating with the user, the agent can offer discount, but they can at most give 10% discounts.

Simulated Users and Data Split. We use Qwen2.5-14B-Instruct, a more powerful model, to simulate the user because it must roleplay as user with significantly different constraints and preferences. Each user has begins with an ideal car (that often does not exist in the database), and they gradually reveal more information about their hard constraints as the agent talks with the user.

The training set and the validation set has the same types of users with randomly generated ideal car and budget. The validation set uses a different set of car brand and car type compared to the training set. The user types are:

1. They will only buy a car that matches their ideal car's brand and type. They require the seller to give them a least one discount, but they are ok with going slightly above budget.
2. They will only buy a car that has at least one of the features in their ideal car. They must be under budget. They are impatient during negotiation, and they will terminate the conversation immediately if the seller takes too long finding a car/price they like.
3. They will only buy a car that matches their ideal car's brand and type. They are flexible with their budget. They are distrustful, so they will never accept the first car suggested by the seller.

The test set has a different set of users with the following types:

1. They will only buy a car that matches their ideal car's brand. They are flexible with their budget. They are impatient during negotiation, and they will terminate the conversation immediately if the seller takes too long finding a car/price they like.
2. They will only buy a car that has at least two of the features in their ideal car. They must be under budget. They are distrustful, so they will never accept the first car suggested by the seller.
3. They will only buy a car that matches their ideal car's type. They want an expensive car. They are impatient with how many cars the seller show to them, and they will terminate the conversation immediately if the seller takes too long.

Reward Function. All non-terminating steps receive a reward of 0.

If the buyer agrees to buy a car, the reward function first verify that the transaction is valid:

1. The car sold is in the database.
2. The seller has not offered over 10% discount.

3. The car satisfies all the buyer's preferences/constraints.

If the transaction is invalid, the agent receives a reward of 0. Otherwise, if the transition is valid, the reward is $(\texttt{purchase\_cost})^2/(\texttt{budget} \times \texttt{market\_price})$.

If the buyer has not agreed to buy anything after the final step, the agent receives a negative reward of $-(\texttt{budget} - \texttt{market\_price})/\texttt{market\_price}$.

G.2 Agent Prompt

G.2.1 API Call

```
{% if mode == 'input' %}
You are roleplaying as a seller in a car dealership. You are talking to a
↪ buyer, and your objective is to call the database APIs so that you can
↪ get information from the database to answer the user's question.

### Car information
Here is all the possible car brands in the database:
{{ all_car_brands }}

Here is all the possible car types in the database:
{{ all_car_types }}

Here is all the possible car features in the database:
{{ all_car_features }}

### API calls that you can use
All the API calls (except no-op) will find cars that satisfy your specified
↪  criteria. Each car will have information about its brand, type,
↪ features, and estimated car price (msrp). You MUST specify the criteria
↪ in the following format:
1. search_car_by_brand_type
This will search for all the cars that satisfy both the "API BRAND" and the
↪  "API TYPE" in the database.

API NAME:
search_car_by_brand_type
API BRAND:
{ TODO: name of the brand that you want to search for }
API TYPE:
{ TODO: name of the type that you want to search for }
API FEATURES:
[]

2. search_car_by_brand
This will search for all the cars that satisfy the "API BRAND" in the
↪ database.

API NAME:
search_car_by_brand
API BRAND:
{ TODO: name of the brand that you want to search for }
API TYPE:
None
API FEATURES:
[]

3. search_car_by_type
```

```
This will search for all the cars that satisfy the "API TYPE" in the
↪ database.

API NAME:
search_car_by_type
API BRAND:
None
API TYPE:
{ TODO: name of the type that you want to search for }
API FEATURES:
[]

4. search_car_that_have_features
This will search for all the cars that have the features in the database.
↪ Features must be a list of strings with double quotes around each
↪ feature.

API NAME:
search_car_that_have_features
API BRAND:
None
API TYPE:
None
API FEATURES:
["feature1", "feature2", ...]

4. no_op
If you don't need any information from the database, you can do nothing.

API NAME:
no_op
API BRAND:
None
API TYPE:
None
API FEATURES:
[]

### Your goal and instructions
Your goal is to use database API to get information about cars in your
↪ dealership. You will decide what API to call based on the chat history
↪ and the previous API call that you have done.

Please follow these general instructions:
* You MUST follow the API call format above.
* You MUST ask about specific brand and/or type of car based on the buyer's
↪  request in the chat history.
* If the previous api request and response already answer the buyer's
↪ request (e.g., the buyer is just negotiating prices), you can choose
↪ no_op since you already have all the necesarry information.
* If the buyer is asking for a brand of car, you can choose
↪ search_car_by_brand.
* If the buyer is asking for a type of car, you can choose
↪ search_car_by_type.
* If the buyer is asking for a car that has certain features, you can
↪ choose search_car_that_have_features. Note that this will show all the
↪ cars that have these features, and they might contain other features as
↪ well.
```

```
* If you are not able to find a car that have all the features that the
↪ buyer wants, you MUST choose to search on a subset of the features
↪ mentioned by the buyer, still using the search_car_that_have_features
↪ API.
* If the buyer has not made any request it, you can choose no_op.
* DO NOT repeat the exact same API call as the previous one. It will NOT
↪ lead to a better outcome.

Below is the history of the conversation so far:
{{ observation_action_history }}

All previous {{ past_N }} API calls (include no-ops):
{{ prev_api_call_history }}

Previous API call (that is not no-op):
{{ previous_api_call }}
Database's response to the previous API call:
{{ previous_api_response }}

### Output format (You MUST ALWAYS have 4 fields: REASON, API NAME, API
↪ BRAND, API TYPE, API FEATURES. You MUST begin directly with the REASON
↪ field)
REASON:
Rationale for what API call to make based on the previous history. In your
↪ reason, consult the list of API calls that you can make to precisely
↪ state VERBATIM what you will do.
API NAME:
name of the API call that you will make
API BRAND:
brand of the car that you will search for (else None)
API TYPE:
type of the car that you will search for (else None)
API FEATURES:
A list of features of the car that you will search for. Each feature must
↪ be a string with double quotes around it. (else [])
{% elif mode == 'output' %}
REASON:
{{ reason }}
API NAME:
{{ api_name }}
API BRAND:
{{ api_brand }}
API TYPE:
{{ api_type }}
API FEATURES:
{{ api_features }}
{% endif %}
```

G.2.2 Reply to Buyer

```
{% if mode == 'input' %}
You are roleplaying as a seller in a car dealership. You are talking to a
↪ buyer, and your objective is to get the buyer to buy the car from you
↪ with as high price as possible.

### Car information
Here is all the possible car brands in the database:
{{ all_car_brands }}
```

```
Here is all the possible car types in the database:
{{ all_car_types }}

### Your goal and instructions
You goal is to (1) You need to output the response to talk to the buyer,
↪ which can be getting information about what type of car they want,
↪ discussing what car your dealership has, or negotiating the price of the
↪  car. (2) You need to select one car from the list of cars that you look
↪  up that you are proposing to the buyer.

Please follow these general instructions:
* You MUST pay close attention to the chat history and what you looked up
↪ in the database (if any) to decide what to say to the buyer.
* If you have not looked up any car yet (maybe you just started the
↪ conversation), you should ask the buyer what type of car they want.
* If you have already looked up some cars, you should discuss ONLY ONE car
↪ with the buyer. Ideally, this should be the most expensive car that
↪ satisfy the buyer's request. In addition to writing down the car index,
↪ you MUST also copy down the car information and write it in the `
↪ PROPOSED CAR BRAND`, `PROPOSED CAR TYPE`, `PROPOSED CAR FEATURES`, and `
↪ PROPOSED CAR MSRP` fields.
* You MUST NOT make up a car that is not in the database. You MUST only
↪ mention cars that are in the database.
* Some car brand do not have the type of car that the buyer wants. If that
↪ happens, you should suggest the type of car that the brand has or
↪ suggest another brand.
* Unless the buyer explicitly asks for a discount, you should NOT offer a
↪ discount. You should just state the market price.
* If the buyer asks for a discount, you MUST start by a discount that is
↪ less than 10% discount (for example, 2% discount). If the buyer is not
↪ satisfied, you can try offering a better discount. You should aim to
↪ offer as little discount as possible because you want to maximize your
↪ profit. The maximum discount you can offer is 10% discount.
* If the discounted car is still not satisfying the buyer's request, you
↪ can try suggesting another similar car.
* Your dealership only has the cars that are in the database. You cannot
↪ add additional features to the car.

### Input
Below is the history of the conversation so far:
{{ observation_action_history }}

API call (that is not no-op):
{{ api_call }}
Database's response to the API call:
{{ api_response }}

Here is the most recent message from the buyer:
{{ buyer_response }}

### Output format (You MUST always have 3 fields: REASON, RESPONSE, CAR
↪ INDEX, PROPOSED CAR BRAND, PROPOSED CAR TYPE, PROPOSED CAR FEATURES,
↪ PROPOSED CAR MSRP. You MUST begin directly with the REASON field)
REASON:
Rationale for what reply you will give to the buyer and which car you will
↪ propose.
RESPONSE:
Your 1-5 sentence response to the buyer.
CAR INDEX:
```

```
The index of the car you are proposing to the buyer (0 if you haven't
↪ looked up any car yet).
PROPOSED CAR BRAND:
Copy down the brand of the car you are proposing to the buyer via the CAR
↪ INDEX. (None if you haven't looked up any car yet)
PROPOSED CAR TYPE:
Copy down the type of the car you are proposing to the buyer via the CAR
↪ INDEX. (None if you haven't looked up any car yet)
PROPOSED CAR FEATURES:
Copy down the features of the car you are proposing to the buyer via the
↪ CAR INDEX. MUST be a list of strings. ([] if you haven't looked up any
↪ car yet)
PROPOSED CAR MSRP:
Copy down the msrp (and integer) of the car you are proposing to the buyer
↪ via the CAR INDEX. (0 if you haven't looked up any car yet)
{% elif mode == 'output' %}
REASON:
{{ reason }}
RESPONSE:
{{ response }}
CAR INDEX:
{{ car_idx }}
PROPOSED CAR BRAND:
{{ proposed_car_brand }}
PROPOSED CAR TYPE:
{{ proposed_car_type }}
PROPOSED CAR FEATURES:
{{ proposed_car_features }}
PROPOSED CAR MSRP:
{{ proposed_car_msrp }}
{% endif %}
```

G.3 Summary Prompt

```
{% if system %}
You are an intelligent assistant summarizing a conversation between a
↪ seller and a buyer, where the seller is trying to sell a car to the
↪ buyer.

## Overall information about the game
Here is all the possible car brands in the database:
{{ all_car_brands }}

Here is all the possible car types in the database:
{{ all_car_types }}

## What you receive as input
You are given (1) a history of what the seller has said and done and (2)
↪ failure reason if the seller failed to sell the car. At each step of the
↪  conversation, the seller first look up a car in the dataset. Then, the
↪ seller will examine what is available in the dataset and potentially
↪ suggest a car to the buyer based on the buyer's request.

## Your goal and rules to follow
You must summarize the chat history by answering the questions below.
1. What kind of requirements are not negotiable for the buyer? What does
↪ the buyer care about the most? You MUST not ignore what the buyer said
↪ in the 1st step because they are just describing their ideal car. You
```

```
↪ MUST pay attention to what the buyer keeps repeating in the remaining
↪ conversation as those requirements are non-negotiable. If features are
↪ not negotiable, did the seller want at least some features or all the
↪ features?
2. What is the mood of buyer? Are they neutral, excited, impatient, etc.?
↪ Were they rushing the negotiation? Were they dubious about the car?

3. What type of API calls are made? Why did the seller make these API calls
↪  based on the conversation history? What are the results of the API
↪ calls?
4. If the API is searching for car based on features ‘
↪ search_car_that_have_features‘, and ‘api_features‘ contains multiple
↪ features, the API will look for cars that have ALL the listed features.
↪ Did the seller just look up one feature, or multiple features? You MUST
↪ be very specific what features the seller was looking for.

5. What type of car have the seller suggested to the buyer? If the API does
↪  not find any car, there will be no car under "Car in the database"
↪ section. You MUST note down if the API call fails to find any car.
↪ However, the seller might make up a car that is not in the database
↪ under the "Copied car information" section. If the seller suggest a car
↪ that is not in the database, you must include that in your summary.
6. Did the seller suggest the most expensive car that satisfies the buyer's
↪  requirements? Did the seller offers a discount?
7. When the seller reject a car suggested by the seller, what is the reason
↪ ?

8. What happened at the last conversation step? Did the buyer end the
↪ negotiation? If so, what is the reason? Was there any car found in the
↪ database? It is problematic if the "Car suggested in the database"
↪ section is empty.

9. If the final transaction is successful, what is the price of the car?
10. If the final transaction is not successful, you must carefully explain
↪ what the seller said at the end, and you might also examine the failure
↪ reason and include that in your summary.

## Output format
You must generate the summary as the following markdown format. You must
↪ answer the questions above in the right section, and you can add other
↪ information that you think is important.
### Summary of the buyer
{ You MUST include your answer to 1. and 2. here. }

### Summary of the API calls
{ You MUST include your answer to 3. and 4. here. }

### Summary of the car suggested
{ You MUST include your answer to 5. and 6. and 7. here}

### Summary of what happened at the last step
{ You MUST include your answer to 8. here. }

### Summary of the overall transaction (success or failure)
{ You MUST include your answer to 9. and 10. here. }
{% endif %}
{% if not system %}
{% if mode == 'input' %}
## Chat History
```

```
{{ chat_history }}

## Failure Reason (if any)
{{ failure_reason }}
{% endif %}
{% endif %}
```

G.4 Hindsight Proposer Prompt

G.4.1 API Call

```
{% if system %}
You are an intelligent teacher who gives guidance on a seller who is trying
↪  to look up a car in the database in response to conversation with a
↪ buyer. The seller only has 10 steps to sell the car to the buyer.

## What you receive as input
You are given (1) a list of car brands, types, and features in the
↪ dealership; (2) the chat history of between the buyer and the seller;
↪ (3) the last {{ past_N }} API calls made by the seller and whether the
↪ API calls have found any car in the database; (4) the most recent no-op
↪ API call, which affects what car the seller can propose to the buyer.

To further help you make wise judgements and provide helpful guidance to
↪ the player, you are also given (5) a summary of the complete interaction
↪  between the buyer and the seller (which include what the buyer cares
↪ about, what API has the seller made, what car the seller has proposed,
↪ and whether the seller has successfully sold the car to the buyer in the
↪  end.

## Your goal and rules to follow
Your goal is to use your hindsight reasoning ability to generate {{
↪ num_responses}} alternative API calls that the seller should have made
↪ given the current chat history. Your process is to: 1. reason about all
↪ the information (including the summary); 2. generate some API calls that
↪  the seller could have made; 3. generate some plausible reasoning that
↪ the seller could have come up with based on the current chat history.

### API calls that the seller can make
All API calls are in the json format. All the API calls (except no-op) will
↪  find cars that satisfy your specified criteria. Each car will have
↪ information about its brand, type, features, and estimated car price (
↪ msrp).
1. search_car_by_brand_type
This will search for all the cars that satisfy both the ‘api_brand‘ and the
↪  ‘api_type‘ in the database. You must format you API call as the
↪ following:
‘‘‘json
{
    "api_name": "search_car_by_brand_type",
    "api_brand": "{ TODO: name of the brand that you want to search for }",
    "api_type": "{ TODO: name of the type that you want to search for }",
    "api_features": [] # THIS MUST BE EMPTY
}
‘‘‘
2. search_car_by_brand
This will search for all the cars that satisfy the ‘api_brand‘ in the
↪ database. You must format you API call as the following:
‘‘‘json
```

````
{
    "api_name": "search_car_by_brand",
    "api_brand": "{ TODO: name of the brand that you want to search for }",
    "api_type": "",
    "api_features": [] # THIS MUST BE EMPTY
}
```
3. search_car_by_type
This will search for all the cars that satisfy the `api_type` in the
↪ database. You must format you API call as the following:
```json
{
    "api_name": "search_car_by_type",
    "api_brand": "",
    "api_type": "{ TODO: name of the type that you want to search for }",
    "api_features": [] # THIS MUST BE EMPTY
}
4. search_car_that_have_features
This will search for all the cars that have the features in the database.
↪ You must format you API call as the following:
```json
{
    "api_name": "search_car_that_have_features",
    "api_brand": "",
    "api_type": "",
    "api_features": ["feature1", "feature2", ...]
}
```
4. no_op
If you don't need any information from the database, you can do nothing.
```json
{
    "api_name": "no_op",
    "api_brand": "",
    "api_type": "",
    "api_features": []
}
```

### Rules that you must follow
- **Make feasible API calls:** The API call that you make MUST be a
↪ feasible API call that the seller can make given the current chat
↪ history.
- **In teacher_reason, reason with all the information:** You are the
↪ teacher. In your teacher reasoning, you MUST make reference what is
↪ discussed in the chat history. You MUST generate API calls that are
↪ possible and reasonable to make given the current chat history. You can
↪ make use of the "### Summary of the Entire Negotation" to help you
↪ identify better but STILL FEASIBLE API calls to make given the current
↪ chat history.
- **In seller_reason, reasoning should not include secret information from
↪ the summary:"** You are generating what is the possible reasoning that a
↪  seller can have based on the chat history in order to generate the API
↪ call. It must be a feasible reasoning based on the chat history, all
↪ previous {{past_N}} API calls, Previous API call, and the "Database's
↪ response to the previous API call". You MUST NOT talk about the summary,
↪  which the seller does not have access to.
````

```
Here are some information about the dealership to help you make better API
↪ calls:
- You must always make sure that the "Database's reseponse to the previous
↪ API call" is not empty. Even if you have made the same API call in "All
↪ previous {{ past_N }} API calls", if the "Database's response to
↪ previous API call" is currently empty, you must make the same API call
↪ that would give you the right set of cars requested by the buyer in the
↪ chat history.
- In the chat history, if the buyer has revealed what are their non-
↪ negotiable requirements (you can get help from the summary), you should
↪ make the API call to search for calls that satisfy the buyer's non-
↪ negotiable requirements. However, in the seller_reason, you MUST make
↪ sure that you only talk about what the seller discussed in the chat
↪ history, not the summary.
- If "Database's response to the previous API call" has cars matching the
↪ buyer's non-negotiable requirements, you can just make a no-op API call.
- Sometimes, "Database's response to the previous API call" might have too
↪ many cars. You can try to make a `search_car_by_brand_type` API call to
↪ narrow down the set of cars.
- `search_car_that_have_features` will return cars that have ALL the
↪ features in the `api_features` list. If there are no cars that have all
↪ the features, you can try searching FOR A SUBSET of the features.

## Output format
The output is a list of JSON containing {{num_responses}} different pairs
↪ of reasoning and feasible API calls that you would have made.
```json
[
    {
        "teacher_reason": "string: your rationale for the API call that you
        ↪ think the seller should have made instead.",
        "api_call": {
            "api_name": "string: the name of the API call that you will make
            ↪ ",
            "api_brand": "string: the brand of the car that you will search
            ↪ for (if applicable)",
            "api_type": "string: the type of the car that you will search
            ↪ for (if applicable)",
            "api_features": ["string: the features that you will search for
            ↪ (if applicable)"]
        },
        "seller_reason": "string: If you are the seller who does not know
        ↪ the entire negotiation history, what will be your reasoning in
        ↪ order to generate the API call based on the chat history alone?
        ↪ You MUST only refer to the chat history, 'All previous {{ past_N
        ↪ }} API calls (include no-ops)', 'Previous API call (that is not
        ↪ no-op)', and 'Database's response to the previous API call'. You
        ↪ MUST NOT talk about content in the summary, which the seller does
        ↪  not have access to. ",
    }
    ...
]
```

Overall information about the current negotiation
Here is all the possible car brands in the database:
{{ all_car_brands }}

Here is all the possible car types in the database:
```

```
{{ all_car_types }}

Here is all the possible car features in the database:
{{ all_car_features }}

## Summary of the Entire Negotiation
{{ summary }}
{% endif %}
{% if not system %}
{% if mode == 'input' %}
Here is the chat history until step {{step_idx}}:
{{ observation_action_history }}

All previous {{ past_N }} API calls (include no-ops):
{{ prev_api_call_history }}

Previous API call (that does not return empty response):
{{ previous_api_call }}
Database's response to the previous API call (This affects what car the
↪ seller can propose to the buyer, so it is important to try to keep this
↪ non-empty if possible):
{{ previous_api_response }}

At this point in the chat history, what API call would you have made? You
↪ MUST generate {{num_responses}} API calls. The API calls should try to
↪ be diverse, but it is ok if sometimes they are the same due to the chat
↪ history.
{% endif %}
{% endif %}
```

G.4.2 Rely to Buyer

```
{% if system %}
You are an intelligent teacher who gives guidance on a seller who negotiate
↪  with a buyer and propose a car that they can buy. The seller only has
↪ 10 steps to sell the car to the buyer.

## What you receive as input
You are given (1) a list of car brands, types in the dealership; (2) the
↪ chat history of between the buyer and the seller; (3) the most recent
↪ API call with non-empty response, which affects what car the seller can
↪ propose to the buyer.

To further help you make wise judgements and provide helpful guidance to
↪ the player, you are also given (4) a summary of the complete interaction
↪  between the buyer and the seller (which include what the buyer cares
↪ about, what API has the seller made, what car the seller has proposed,
↪ and whether the seller has successfully sold the car to the buyer in the
↪  end).

## Your goal and rules to follow
Your goal is to use your hindsight reasoning ability to generate {{
↪ num_responses}} alternative responses and proposed car (if applicable)
↪ that the seller should have made given the current chat history. Your
↪ process is to: 1. reason about all the information (including the
↪ summary); 2. generate some responses and proposed car that the seller
↪ could have made; 3. generate some plausible reasoning that the seller
↪ could have come up with based on the current chat history.
```

```
### Rules that you must follow
- **Generate feasible responses:** The response that you generate MUST be a
↪  feasible response that the seller can make given the current chat
↪ history. You MUST NOT make up a car that is not in the database in the
↪ response.
- **Select feasible proposed car:** If "Database's response to the previous
↪  API call" is not empty, you CAN select the index of a car from the list
↪  to propose to the buyer. However, if that field is empty, you MUST
↪ leave the car_index as 0.
- **Copy down the proposed car information:** If you selected a car (which
↪ CAN ONLY happen if "Database's response to the previous API call" is not
↪  empty", you MUST copy down the information of the car that you are
↪ proposing to the user. You MUST NOT make up a car that is not in the
↪ database.
- **In teacher_reason, reason with all the information:** You are the
↪ teacher. In your teacher reasoning, you MUST make reference to what is
↪ discussed in the chat history. You MUST generate responses that are
↪ possible and reasonable to make given the current chat history. You can
↪ make use of the "### Summary of the Entire Negotation" to help you
↪ identify better but STILL FEASIBLE responses to make given the current
↪ chat history.
- **In seller_reason, reasoning should not include secret information from
↪ the summary:"** You are generating what is the possible reasoning that a
↪  seller can have based on the chat history in order to generate the
↪ response, car_idx, and proposed_car. It must be a feasible reasoning
↪ based on the chat history alone.

Here are some information about the dealership to help you make better
↪ responses and select better cars:
- You MUST NOT make a car that is not in the database.
- You can ONLY propose a car to the user if "Database's response to the
↪ previous API call" is not empty. You MUST NOT make up general claims
↪ about the car that is not in the database.
- When you select a car, ideally, you should select the most expensive car
↪ that satisfy the buyer's request.
- If the buyer asks for a discount, you MUST start by a discount that is
↪ less than 10% discount (for example, 2% discount). If the buyer is not
↪ satisfied, you can try offering a better discount. You should aim to
↪ offer as little discount as possible because you want to maximize your
↪ profit. The maximum discount you can offer is 10% discount.
- Once you have exhausted the maximum discount, you MUST try suggesting
↪ another car based on what is the buyer's non-negotiable requirements
↪ based on the chat history (and you can also get help from the summary).
- You MUST NOT get stuck in a loop of suggesting the same car over and over
↪  again especially if you have already given the maximum discount and the
↪  buyer is asking for a cheaper price again.

## Output format
The output is a list of JSON containing {{num_responses}} different pairs
↪ of reasoning and feasible responses that you would have made.
'''json
[
    {
        "teacher_reason": "string: your rationale for the response that you
        ↪ think the seller should have made instead.",
        "response": "string: your 1-5 sentence response to the buyer",
        "car_idx": "integer: the index of the car you are proposing to the
        ↪ buyer (0 if you haven't looked up any car yet)",
        "proposed_car": {
```

```
            "brand": "string: the brand of the car. Empty string if you
            ↪ haven't looked up any car yet",
            "type": "string: the type of the car. Empty string if you haven'
            ↪ t looked up any car yet",
            "features": ["string: the features of the car. Empty list if you
            ↪  haven't looked up any car yet",
            "msrp": "integer: the manufacturer's suggested retail price of
            ↪ the car. 0 if you haven't looked up any car yet"
        }
        "seller_reason": "string: If you are the seller who does not know
        ↪ the entire negotiation history, what will be your reasoning in
        ↪ order to generate the response, car_idx, and proposed_car based
        ↪ on the chat history alone? You MUST only refer to the chat
        ↪ history, 'Previous API call (that is not no-op)', and 'Database's
        ↪  response to the previous API call'. You MUST NOT talk about
        ↪ content in the summary, which the seller does not have access to.
        ↪  ",
    }
    ...
]
'''

## Overall information about the current negotiation
Here is all the possible car brands in the database:
{{ all_car_brands }}

Here is all the possible car types in the database:
{{ all_car_types }}

## Summary of the Entire Negotiation
{{ summary }}
{% endif %}
{% if not system %}
{% if mode == 'input' %}
Here is the chat history until step {{step_idx}}:
{{ observation_action_history }}

Previous API call (that does not return empty response):
{{ api_call_used }}
Database's response to the previous API call (This affects what car the
↪ seller can propose to the buyer, so it is important to try to keep this
↪ non-empty if possible):
{{ api_response_used }}

At this point in the chat history, what response would you have made? You
↪ MUST generate {{num_responses}} responses. The responses should try to
↪ be diverse.
{% endif %}
{% endif %}
```