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

Huaxiaoyue Wang [1] **Sanjiban Choudhury** [1]

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 (Ouyang et al., 2022; Rafailov et al., 2023; Stiennon et al., 2020) and enhancing their reasoning capabilities (DeepSeek-AI et al., 2025; Lightman et al., 2024; Uesato et al., 2022; Wang et al., 2024). Recent works apply RL to train LLM agents for multi-turn decision-making tasks, such as code generation (Jain et al., 2025; Le et al., 2022), web navigation (Bai et al., 2024; Putta et al., 2024; Qi et al., 2025), and task-oriented dialogues (Abdulhai et al., 2023; Verma et al., 2022; Zhou et al., 2024), where each action by the agent influences future states in the environment.

However, exploration remains a key challenge in RL (Brafman & Tennenholtz, 2002; Strens, 2000; Thrun, 1992). Classical exploration strategies that sample random actions, such as ϵ-greedy and upper confidence bound (Auer et al., 2002), 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 (Bai et al., 2022; Kim et al., 2023; Madaan et al., 2023; Shinn et al., 2023), 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. 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

[1]Cornell University. Correspondence to: Huaxiaoyue (Yuki) Wang <yukiwang@cs.cornell.edu>.

ICML 2025 Workshop on Efficient Systems for Foundation Models (ES-FoMo III), Copyright 2025 by the author(s).

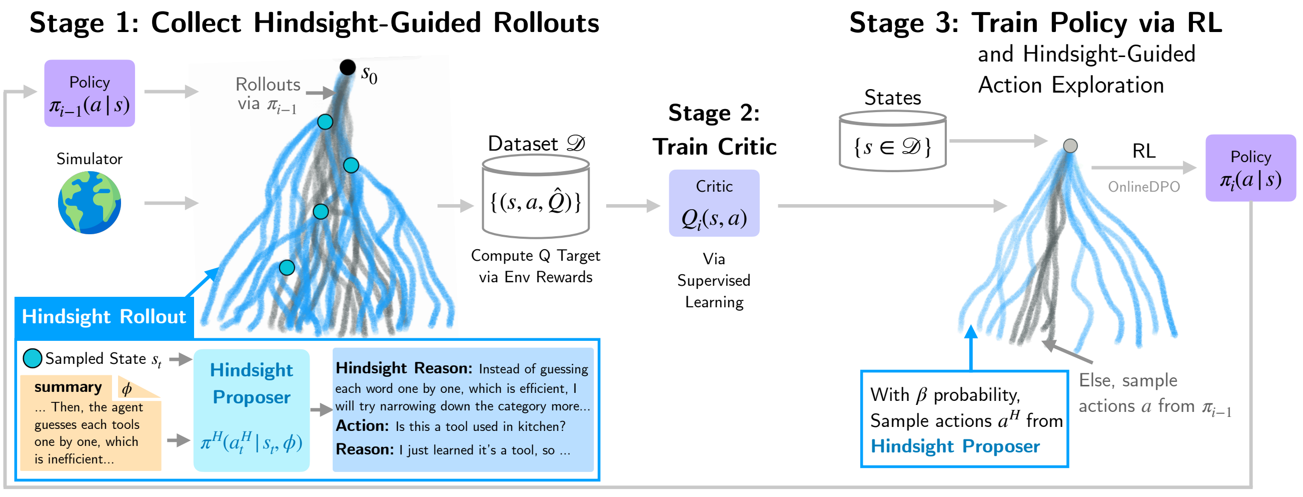
Stage 1: Collect Hindsight-Guided Rollouts
Policy $\pi_{i-1}(a \mid s)$
Simulator
Rollouts via π_{i-1}
s_0
Dataset $\mathcal{D}$
$\{(s, a, \hat{Q})\}$
Compute Q Target via Env Rewards
Hindsight Rollout
Sampled State s_t
summary ϕ
... Then, the agent guesses each tools one by one, which is inefficient...
Hindsight Proposer
$\pi^H(a_t^H \mid s_t, \phi)$
Hindsight Reason: Instead of guessing each word one by one, which is efficient, I will try narrowing down the category more...
Action: Is this a tool used in kitchen?
Reason: I just learned it's a tool, so ...
Stage 2: Train Critic
Critic $Q_i(s, a)$
Via Supervised Learning
Stage 3: Train Policy via RL and Hindsight-Guided Action Exploration
States
$\{s \in \mathcal{D}\}$
RL
OnlineDPO
Policy $\pi_i(a \mid s)$
With β probability, Sample actions a^H from Hindsight Proposer
Else, sample actions a from π_{i-1}

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.

exploration into 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 (Abdulhai et al., 2023), showing that HOPE outperforms leading multi-turn RL approaches on task-oriented dialogue and tool-use tasks.

2. 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 2.1), which are then used to improve state-action coverage during critic training (Section 2.2) and steer exploration during actor training (Section 2.3).

2.1. 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 $\tau = (s_0, a_0, r_0, \dots)$. 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 (Bai et al., 2022; Kim et al., 2023; Madaan et al., 2023; Shinn et al., 2023) or generating post-hoc explanations conditioned on ground-truth answers (Wadhwa et al., 2024; Xu et al., 2024; Zelikman et al., 2022). 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.

2.2. 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

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

```
1: Initialize with agent policy $\pi_0$
2: for iteration $i = 1, \ldots, K$ do
3:     // Stage 1: Collect rollouts (Explore with HOPE)
4:     Collect on-policy rollouts $\{(\ldots, s_t, a_t, r_t, \ldots)\}$ using $\pi_{i-1}$ and store in replay buffer $\mathcal{G}(s,a)$
5:     Collect hindsight-guided rollouts Algorithm 2 and maintain data ratio $\alpha$ in $\mathcal{G}(s,a)$
6:     // Stage 2: Train Process Reward Model
7:     Compute PRM targets via (3) and aggregate into dataset $\mathcal{D} = \{(s, a, \hat{Q})\}$
8:     Train PRM $Q_i$ to minimize soft binary cross-entropy loss:
           $$Q_i = \arg\min_Q -\mathbb{E}_{(s,a,\hat{Q})\sim\mathcal{D}} \left[\hat{Q} \log Q(s,a) + (1-\hat{Q}) \log(1-Q(s,a))\right] \quad (1)$$
9:     // Stage 3: Train Policy via RL (Explore with HOPE)
10:    Train policy $\pi_i$ to maximize $Q_i$, exploring with probability $\beta$ via hindsight proposer $\pi^H$
           $$\pi_i = \arg\max_\pi \mathbb{E}_{s\sim\mathcal{D}, a\sim\text{Sample}(s,\pi_i,\pi^H,\beta)} [Q_i(s,a)] - \beta\mathcal{D}_{\text{KL}} [\pi(a \mid s) \| \pi_{i-1}(a \mid s)] \quad (2)$$
11: end for
12: return Best $\pi \in \{\pi_1, \ldots, \pi_K\}$ on validation dataset
```

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 $\{\tau\}$.

2.3. 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 (Xie et al., 2024; Zhang et al., 2025).

3. Experiments

We discuss domain, training details, and additional experiments that (1) compare HOPE against other unguided/guided exploration strategies Section E.1 and (2) study the optimal data mixture for critic training Section E.2 in the appendix.

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 (Grattafiori et al., 2024) (denoted as **3B**). Following RLHF (Ouyang et al., 2022), 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** (Choudhury & Sodhi, 2025) 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. 2.1). For RL, we compare with Rejection Sampling (**Reject-S** for short) (Grattafiori et al., 2024) 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.

3.1. 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 roll-

		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. HOPE consistently outperforms other IL and RL baselines.

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.

outs 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. Finally, the IL baseline LEAP often underperforms, as it simply imitates the hindsight proposer, which may generate suboptimal or infeasible actions in certain contexts.

3.2. 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.

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 $(0.55 \to 0.77)$. When a critic is ineffective $(\alpha = 0)$, guided exploration during actor training can boost performance $(0.55 \to 0.62)$. When the critic is trained with a sufficient state-action coverage, additional exploration in actor training does not yield additional gains.

4. Discussion

We present HOPE, which leverages the hindsight reasoning capabilities of LLMs to guide exploration in multi-turn reinforcement learning. Rather than directly imitating counterfactual actions, HOPE integrates them into both actor and critic training. By combining rewards with a learned critic, the method identifies which counterfactuals lead to better outcomes and uses them to shape future behavior. Still, a few limitations remain: (1) HOPE inserts a single counterfactual action per trajectory before reverting to on-policy rollouts, as full rollouts from counterfactuals require expensive environment simulation; and (2) experiments were conducted with LLMs up to three billion parameters, leaving open the question of how well these findings generalize to larger models. Looking ahead, we see HOPE as a step toward more efficient reinforcement learning with foundation models—where learning is guided not just by trial and error, but by reasoning informed over past experience.

References

Abdulhai, M., White, I., Snell, C., Sun, C., Hong, J., Zhai, Y., Xu, K., and Levine, S. Lmrl gym: Benchmarks for multi-turn reinforcement learning with language models, 2023. URL https://arxiv.org/abs/2311.18232.

Arumugam, D. and Griffiths, T. L. Toward efficient exploration by large language model agents, 2025. URL https://arxiv.org/abs/2504.20997.

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2–3):235–256, May 2002. ISSN 0885-6125. doi: 10.1023/A:1013689704352. URL https://doi.org/10.1023/A:1013689704352.

Bai, C., Zhang, Y., Qiu, S., Zhang, Q., Xu, K., and Li, X. Online preference alignment for language models via count-based exploration. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=cfKZ5VrhXt.

Bai, H., Zhou, Y., Cemri, M., Pan, J., Suhr, A., Levine, S., and Kumar, A. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 12461–12495. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/1704ddd0bb89f159dfe609b32c889995-Paper-Conference.pdf.

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., Kerr, J., Mueller, J., Ladish, J., Landau, J., Ndousse, K., Lukosuite, K., Lovitt, L., Sellitto, M., Elhage, N., Schiefer, N., Mercado, N., DasSarma, N., Lasenby, R., Larson, R., Ringer, S., Johnston, S., Kravec, S., Showk, S. E., Fort, S., Lanham, T., Telleen-Lawton, T., Conerly, T., Henighan, T., Hume, T., Bowman, S. R., Hatfield-Dodds, Z., Mann, B., Amodei, D., Joseph, N., McCandlish, S., Brown, T., and Kaplan, J. Constitutional ai: Harmlessness from ai feedback, 2022. URL https://arxiv.org/abs/2212.08073.

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

Carta, T., Romac, C., Wolf, T., Lamprier, S., Sigaud, O., and Oudeyer, P.-Y. Grounding large language models in interactive environments with online reinforcement learning. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 3676–3713. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/carta23a.html.

Cen, S., Mei, J., Goshvadi, K., Dai, H., Yang, T., Yang, S., Schuurmans, D., Chi, Y., and Dai, B. Value-incentivized preference optimization: A unified approach to online and offline RLHF. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=SQnitDuow6.

Chen, B., Shu, C., Shareghi, E., Collier, N., Narasimhan, K., and Yao, S. Fireact: Toward language agent finetuning, 2023. URL https://arxiv.org/abs/2310.05915.

Choudhury, S. and Sodhi, P. Better than your teacher: LLM agents that learn from privileged AI feedback. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=st7XqFgbAH.

Coda-Forno, J., Binz, M., Akata, Z., Botvinick, M., Wang, J., and Schulz, E. Meta-in-context learning in large language models. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 65189–65201. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/cda04d7ea67ea1376bf8c6962d8541e0-Paper-Conference.pdf.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Ye, S., Yun, T., Pei, T., Sun, T., Wang, T., Zeng, W., Zhao, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen, X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z., and Zhang, Z. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Dwaracherla, V., Asghari, S. M., Hao, B., and Van Roy, B. Efficient exploration for LLMs. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12215–12227. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/dwaracherla24a.html.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D., Livshits, D., Wyatt, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Guzmán, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Thattai, G., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Zhang, J., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Prasad, K., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Lakhotia, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Tsimpoukelli, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M. K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Zhang, N., Duchenne, O., Çelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Maheswari, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S. S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Albiero, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Wang, X., Tan, X. E., Xia, X., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z. D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Srivastava, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenberg, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Teo, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Dong, A., Franco, A., Goyal, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., Paola, B. D., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock, B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Liu, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Gao, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Le, E.-T., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Kokkinos, F., Ozgenel, F., Caggioni, F., Kanayet, F., Seide, F., Florez, G. M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Inan, H., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Zhan, H., Damlaj, I., Molybog, I., Tufanov, I., Leontiadis, I., Veliche, I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Lam, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J., Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U, K. H., Saxena, K., Khandelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Jagadeesh, K., Huang, K., Chawla, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Liu, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M. J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa, N., Singhal, N., Egebo, N., Usunier, N., Mehta, N., Laptev, N. P., Dong, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Parthasarathy, R., Li, R., Hogan, R., Battey, R., Wang, R., Howes, R., Rinott, R., Mehta, S., Siby, S., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Mahajan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S. C., Patil, S., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Deng, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Koehler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wu, X., Wang, X., Wu, X., Gao, X., Kleinman, Y., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Zhao, Y., Hao, Y., Qian, Y., Li, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., Zhao, Z., and Ma, Z. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Guo, S., Zhang, B., Liu, T., Liu, T., Khalman, M., Llinares, F., Rame, A., Mesnard, T., Zhao, Y., Piot, B., Ferret, J., and Blondel, M. Direct language model alignment from online ai feedback, 2024. URL https://arxiv.org/abs/2402.04792.

Jain, A. K., Gonzalez-Pumariega, G., Chen, W., Rush, A. M., Zhao, W., and Choudhury, S. Multi-turn code generation through single-step rewards. In *Workshop on Reasoning and Planning for Large Language Models*, 2025. URL https://openreview.net/forum?id=B9kbmNtWIp.

Kim, G., Baldi, P., and McAleer, S. M. Language models can solve computer tasks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=M6OmjAZ4CX.

Krishnamurthy, A., Harris, K., Foster, D. J., Zhang, C., and Slivkins, A. Can large language models explore in-context?, 2024. URL https://arxiv.org/abs/2403.15371.

Kroeger, N., Ley, D., Krishna, S., Agarwal, C., and Lakkaraju, H. Are large language models post hoc explainers? In *R0-FoMo:Robustness of Few-shot and Zero-shot Learning in Large Foundation Models*, 2023. URL https://openreview.net/forum?id=1G7n7LW3mF.

Kumar, A., Zhuang, V., Agarwal, R., Su, Y., Co-Reyes, J. D., Singh, A., Baumli, K., Iqbal, S., Bishop, C., Roelofs, R., Zhang, L. M., McKinney, K., Shrivastava, D., Paduraru, C., Tucker, G., Precup, D., Behbahani, F., and Faust, A. Training language models to self-correct via reinforcement learning, 2024. URL https://arxiv.org/abs/2409.12917.

Le, H., Wang, Y., Gotmare, A. D., Savarese, S., and Hoi, S. C. H. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 21314–21328. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/8636419dea1aa9fbd25fc4248e702da4-Paper-Conference pdf.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Liu, Z., Hu, H., Zhang, S., Guo, H., Ke, S., Liu, B., and Wang, Z. Reason for future, act for now: A principled framework for autonomous llm agents with provable sample efficiency, 2024. URL https://arxiv.org/abs/2309.17382.

Lu, J., Dou, Z., Wang, H., Cao, Z., Dai, J., Wan, Y., and Guo, Z. Autopsv: Automated process-supervised verifier, 2024. URL https://arxiv.org/abs/2405.16802.

Luo, L., Liu, Y., Liu, R., Phatale, S., Guo, M., Lara, H., Li, Y., Shu, L., Zhu, Y., Meng, L., Sun, J., and Rastogi, A. Improve mathematical reasoning in language models by automated process supervision, 2024. URL https://arxiv.org/abs/2406.06592.

Ma, K., Zhang, H., Wang, H., Pan, X., and Yu, D. LASER: LLM agent with state-space exploration for web navigation. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023. URL https://openreview.net/forum?id=sYFFyAILy7.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., and Clark, P. Self-refine: Iterative refinement with self-feedback. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Nie, A., Su, Y., Chang, B., Lee, J. N., Chi, E. H., Le, Q. V., and Chen, M. Evolve: Evaluating and optimizing llms for exploration, 2024. URL https://arxiv.org/abs/2410.06238.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. 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. ISBN 9781713871088.

Putta, P., Mills, E., Garg, N., Motwani, S., Finn, C., Garg, D., and Rafailov, R. Agent q: Advanced reasoning and learning for autonomous ai agents, 2024. URL https://arxiv.org/abs/2408.07199.

Qi, Z., Liu, X., Iong, I. L., Lai, H., Sun, X., Sun, J., Yang, X., Yang, Y., Yao, S., Xu, W., Tang, J., and Dong, Y. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=oVKEAFjEqv.

Qu, Y., Zhang, T., Garg, N., and Kumar, A. Recursive introspection: Teaching language model agents how to self-improve. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 55249–55285. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/639d992f819c2b40387d4d5170b8ffd7-Paper-Conference.pdf.

Qwen, :, Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., Lu, K., Bao, K., Yang, K., Yu, L., Li, M., Xue, M., Zhang, P., Zhu, Q., Men, R., Lin, R., Li, T., Tang, T., Xia, T., Ren, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Wan, Y., Liu, Y., Cui, Z., Zhang, Z., and Qiu, Z. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=HPuSIXJaa9.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Setlur, A., Nagpal, C., Fisch, A., Geng, X., Eisenstein, J., Agarwal, R., Agarwal, A., Berant, J., and Kumar, A. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=A6Y7AqlzLW.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: language agents with verbal reinforcement learning. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/

1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Song, Y., Yin, D., Yue, X., Huang, J., Li, S., and Lin, B. Y. Trial and error: Exploration-based trajectory optimization of LLM agents. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7584–7600, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.409. URL https://aclanthology.org/2024.acl-long.409/.

Stiennon, N., Ouyang, L., Wu, J., Ziegler, D. M., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. 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. ISBN 9781713829546.

Strens, M. J. A. A bayesian framework for reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pp. 943–950, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1558607072.

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

Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

Verma, S., Fu, J., Yang, S., and Levine, S. CHAI: A CHatbot AI for task-oriented dialogue with offline reinforcement learning. In Carpuat, M., de Marneffe, M.-C., and Meza Ruiz, I. V. (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4471–4491, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.332. URL https://aclanthology.org/2022.naacl-main.332/.

Wadhwa, S., Amir, S., and Wallace, B. C. Investigating mysteries of CoT-augmented distillation. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 6071–6086, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.349. URL https://aclanthology.org/2024.emnlp-main.349/.

Wang, P., Li, L., Shao, Z., Xu, R., Dai, D., Li, Y., Chen, D., Wu, Y., and Sui, Z. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

Xie, T., Foster, D. J., Krishnamurthy, A., Rosset, C., Awadallah, A., and Rakhlin, A. Exploratory preference optimization: Harnessing implicit q*-approximation for sample-efficient rlhf, 2024. URL https://arxiv.org/abs/2405.21046.

Xu, R., Qi, Z., and Xu, W. Preemptive answer "attacks" on chain-of-thought reasoning. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 14708–14726, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.876. URL https://aclanthology.org/2024.findings-acl.876/.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zelikman, E., Wu, Y., Mu, J., and Goodman, N. STar: Bootstrapping reasoning with reasoning. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_3ELRdg2sgI.

Zhai, Y., Bai, H., Lin, Z., Pan, J., Tong, S., Zhou, Y., Suhr, A., Xie, S., LeCun, Y., Ma, Y., and Levine, S. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 110935–110971. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/c848b7d3adc08fcd0bf1df3101ba6728-Paper-Conference.pdf.

Zhang, S., Yu, D., Sharma, H., Zhong, H., Liu, Z., Yang, Z., Wang, S., Awadalla, H. H., and Wang, Z. Self-exploring language models: Active preference elicitation for online alignment. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=FoQK84nwY3.

Zhou, Y., Zanette, A., Pan, J., Levine, S., and Kumar, A. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=b6rA0kAHT1.

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 (Coda-Forno et al., 2023; Krishnamurthy et al., 2024; Liu et al., 2024; Nie et al., 2024), while others investigate whether LLMs can self-refine and explore actions given feedback (Bai et al., 2022; Kim et al., 2023; Madaan et al., 2023; Shinn et al., 2023). However, prompting-based approaches require significant engineering efforts (e.g., by explicitly modeling the structure of the state space (Ma et al., 2023)).

RLHF (Ouyang et al., 2022) 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 (Cen et al., 2025; Dwaracherla et al., 2024; Xie et al., 2024; Zhang et al., 2025), 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 (Arumugam & Griffiths, 2025; Dwaracherla et al., 2024). Dwaracherla et al. (Dwaracherla et al., 2024) 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 (Arumugam & Griffiths, 2025) 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 (Kroeger et al., 2023; Wadhwa et al., 2024; Xu et al., 2024; Zelikman et al., 2022). 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 (DeepSeek-AI et al., 2025; Lightman et al., 2024; Wang et al., 2024), learning self-correction (Kumar et al., 2024; Qu et al., 2024), code generation (Jain et al., 2025; Le et al., 2022), and interactive environments (Bai et al., 2024; Carta et al., 2023; Putta et al., 2024; Qi et al., 2025; Zhai et al., 2024). A class of approaches utilizes rejection sampling to only fine-tune the LLM on successful trajectories (Chen et al., 2023; Grattafiori et al., 2024; Zelikman et al., 2022). Some works also utilize failed trajectory by learning from expert correction (Choudhury & Sodhi, 2025) or a contrastive loss (Song et al., 2024). Meanwhile, ARCHER (Zhou et al., 2024) 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 (Lightman et al., 2024; Lu et al., 2024; Luo et al., 2024; Setlur et al., 2025; Uesato et al., 2022; Wang et al., 2024). 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. 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 (Lightman et al., 2024; Setlur et al., 2025; Uesato et al., 2022; Wang et al., 2024), 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:

$$\hat{Q}(s, a) = \frac{1}{|\mathcal{G}(s, a)|} \sum_{(s_t, a_t) \in \mathcal{G}(s, a)} \sum_{k=t}^{T-1} \gamma^{k-t} r(s_k, a_k). \tag{3}$$

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 (Schulman et al., 2017), rejection sampling (Grattafiori et al., 2024), or Online DPO (Guo et al., 2024).

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.

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).

Algorithm 2 HOPE Data Collection For Critic

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

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 (Yao et al., 2023) 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.

C.2. Collecting Trajectories

Algorithm 2 shows how HOPE leverage the hindsight proposer to increase state-action coverage for critic training data.

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.

[1]https://github.com/allenai/open-instruct
[2]https://github.com/sgl-project/sglang

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.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.

D. Experiments Setup

Multi-Turn Environments. We evaluate our approach across 3 multi-turn conversational environments proposed in the LMRL Gym benchmark (Abdulhai et al., 2023). 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 (Grattafiori et al., 2024) 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 (Grattafiori et al., 2024) 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 (Qwen et al., 2025). 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 (Grattafiori et al., 2024) (denoted as **3B**). Following RLHF (Ouyang et al., 2022), 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** (Choudhury & Sodhi, 2025) 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. 2.1). For RL, we compare with Rejection Sampling (**Reject-S** for short) (Grattafiori et al., 2024) 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. We perform two iterations for all approaches. Both LEAP and Reject-S trains from the base model (Llama-3.2-3B-Instruct) instead of π_0 for 1 epoch because we empirically observe that training from π_0 degrades performance. In contrast, both PRM+RL and HOPE initialize the actor and critic before training with weights from π_0 because effective RL requires a policy that has already learned to format its output based on prompt instruction. To enable the critic to output scalar Q-values, we append a randomly initialized linear layer (Ouyang et al., 2022; Stiennon et al., 2020), and we train it from a dataset with a fixed size of 10k datapoints. Both approaches optimize the actor using OnlineDPO (Bai et al., 2025). By default, we disable hindsight-guided exploration during actor training, setting $\beta = 0$. We model HOPE's hindsight proposer $\pi^H(a^H|s, \phi)$ via gpt4o. Appendix includes detailed information about the ratio α of hindsight data used to train HOPE's critic in each domain, prompts, and training hyperparameters,

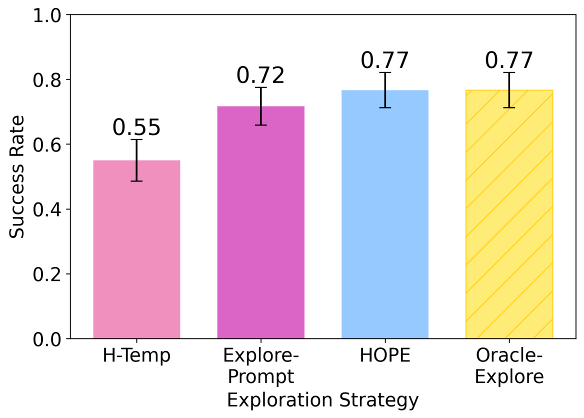

Figure 2. **Policy performance given different exploration strategies.** We use the test set of TwentyQuestions to evaluate π_1 trained with different exploration approaches and report the average success with standard error.

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.

E. Additional Experimental Results

E.1. 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.

						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.

Figure 2 shows that `HOPE` matches the success rate of `Oracle-Explore` (0.77), indicating that the hindsight proposer can identify comparably useful exploratory actions without access to a strong oracle. While less effective, `Explore-Prompt` still outperforms `H-Temp` (0.72 vs. 0.55), suggesting that prompt-level instruction improves over naive temperature sampling. However, since `Explore-Prompt` only encourages superficial diversity—without targeting high-value counterfactuals—it is less effective at uncovering rewarding states and actions compared to `HOPE` and `Oracle-Explore`.

E.2. 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.

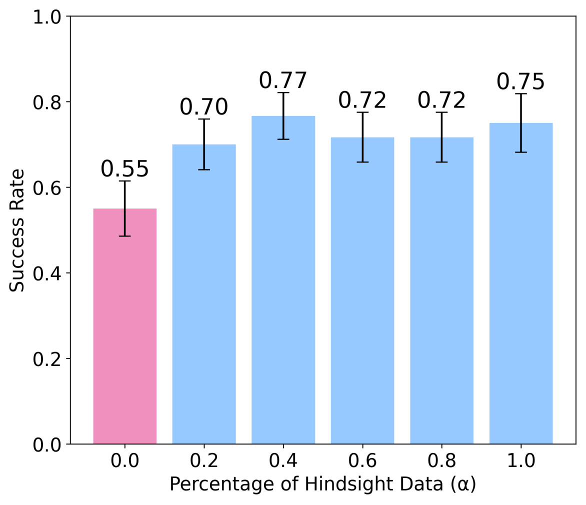

Figure 3. **Effect of critic data mixture on downstream policy performance**. We vary the percentage of hindsight data via the hyperparameter α and train the corresponding critic and $\pi 1$. We report the average success with standard error in TwentyQuestions.

E.3. 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 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.

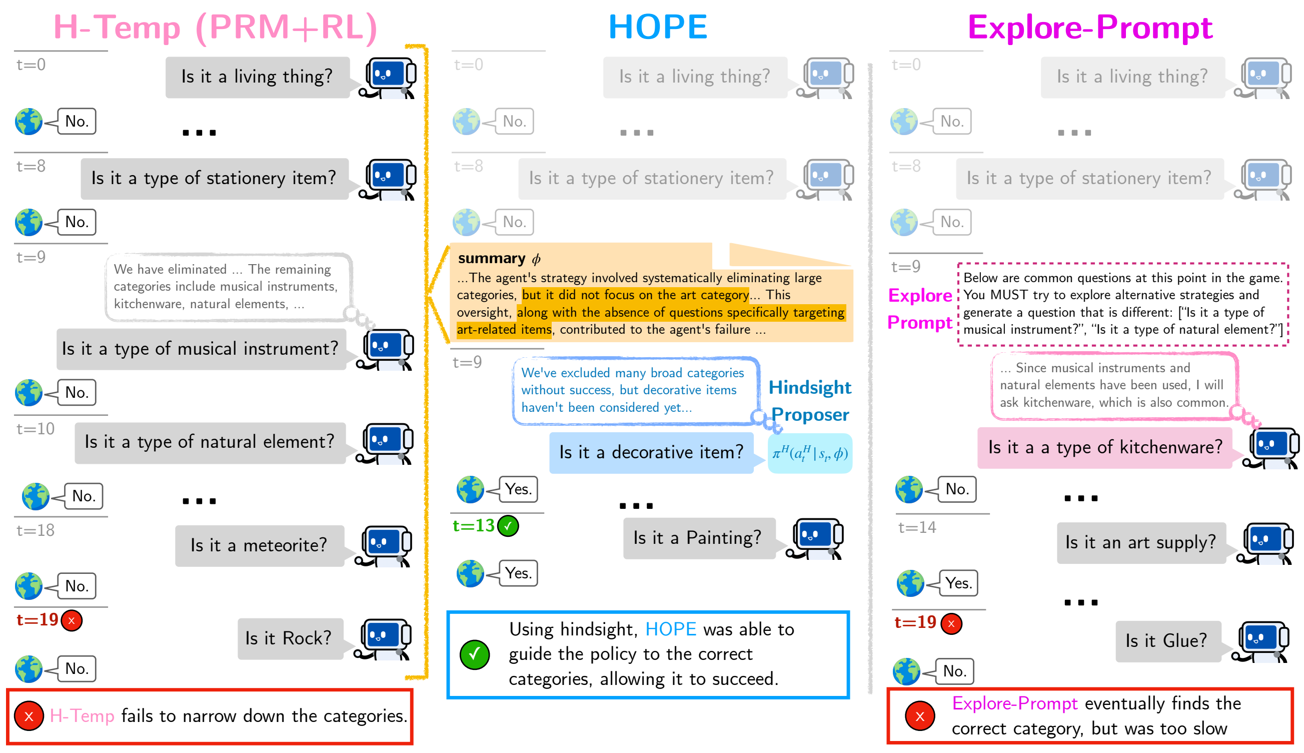
H-Temp (PRM+RL)
HOPE
Explore-Prompt
t=0
Is it a living thing?
No.
t=8
Is it a type of stationery item?
No.
t=9
We have eliminated ... The remaining categories include musical instruments, kitchenware, natural elements, ...
Is it a type of musical instrument?
No.
t=10
Is it a type of natural element?
No.
t=18
Is it a meteorite?
No.
t=19
Is it Rock?
No.
H-Temp fails to narrow down the categories.
summary ϕ
...The agent's strategy involved systematically eliminating large categories, but it did not focus on the art category... This oversight, along with the absence of questions specifically targeting art-related items, contributed to the agent's failure ...
t=9
We've excluded many broad categories without success, but decorative items haven't been considered yet...
Hindsight Proposer
Is it a decorative item?
$\pi^H(a_t^H \mid s_t, \phi)$
Yes.
t=13
Is it a Painting?
Yes.
Using hindsight, HOPE was able to guide the policy to the correct categories, allowing it to succeed.
Explore Prompt
Below are common questions at this point in the game. You MUST try to explore alternative strategies and generate a question that is different: ["Is it a type of musical instrument?", "Is it a type of natural element?"]
... Since musical instruments and natural elements have been used, I will ask kitchenware, which is also common.
Is it a a type of kitchenware?
No.
t=14
Is it an art supply?
Yes.
t=19
Is it Glue?
No.
Explore-Prompt eventually finds the correct category, but was too slow

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.

E.4. 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 $t = 13$.

F. 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.

G. Domain: TwentyQuestions Details

G.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.

G.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 %}
```

G.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 %}
```

G.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 %}
~~~

H. Domain: GuessMyCity Details

H.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.

H.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 %}
```

H.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 %}
```

H.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 %}
```

I. Domain: CarDealer Details

I.1. Environment Setup

We improve the original CarDealer environment in LMRL (Abdulhai et al., 2023) 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 $-(\texttt{budget} - \texttt{market\_price})/\texttt{market\_price}$.

I.2. Agent Prompt

I.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 %}
```

I.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 %}
```

I.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 %}
```

I.4. Hindsight Proposer Prompt

I.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 %}
```

### I.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 %}
```