ALIGNING LLMS TOWARD MULTI-TURN CONVERSATIONAL OUTCOMES USING ITERATIVE RLHF

Daniel R. Jiang*
Meta AI

Ankur Samanta*†
Columbia University
Meta AI

Yukai Yang
Meta AI

Jalaj Bhandari
Meta AI

Rémi Munos
FAIR at Meta

Tyler Lu
Meta AI

ABSTRACT

Training large language models (LLMs) as multi-turn conversational agents remains a significant challenge, particularly in goal-oriented settings. The difficulty stems from sparse, long-horizon objectives and the discrepancy between response-level planning and token-level generation. In this paper, we present a formal reduction of the multi-turn RL problem into a *sequence of single-turn RLHF-style problems*. This is achieved by setting a learned multi-turn Q-function as the reward model for the single-turn problem. We demonstrate and prove a key insight: solving this single-turn RLHF problem with standard token-level approach (e.g., GRPO or PPO) is equivalent to an approximate policy improvement step within the multi-turn problem. This insight naturally leads to *Iterative GRPO*, a batch online approximate policy iteration algorithm that alternates between collecting a batch of data from the current policy, fitting Q-functions from these logged conversation trajectories, and improving the policy via single-turn RLHF. A major practical advantage is that Iterative GRPO directly leverages stable, off-the-shelf single-turn RLHF tools, making it straightforward to implement. Empirically, we demonstrate the effectiveness of Iterative GRPO on new multi-turn conversational environments inspired by sales-oriented agent-customer interactions.

1 INTRODUCTION

There has been significant progress in post-training large language models (LLMs) with reinforcement learning (RL). At the same time, LLMs have driven rapid advances in conversational AI agents (or "chatbots") across many applications, including assistants that aim to drive specific *outcomes* (e.g., resolving a support issue, completing a task, or helping a customer reach a decision). Despite this, most widely used RLHF tooling targets *single-turn* objectives, where feedback is provided directly on an individual response. In contrast, outcome-driven conversations are naturally *multi-turn*: success is often only known after the full interaction ends. This leads to sparse terminal feedback and a long-horizon credit-assignment problem, since earlier responses may matter only through their downstream effects on later turns.

While research in multi-turn RL algorithms is rapidly advancing, training LLMs for multi-turn problems is often more complex in practice, and high-quality, standardized RL implementations are less common than for single-turn RLHF. As a result, there is a methodological mismatch: we want to optimize multi-turn outcomes, but our most mature tools are designed for single-turn training.

In this work, we address this gap by reducing response-level multi-turn RL to a sequence of standard single-turn (token-level) RL problems. The key idea is to learn a value function Q^π under the current policy π that predicts the expected terminal outcome and thereby "compresses" the rest of the conversation into a single-turn reward signal. This yields a simple *batch online* pipeline that

*Equal contribution.
†Work done at Meta.

alternates between (i) collecting multi-turn trajectories and fitting Q^π from observed outcomes and (ii) performing policy improvement using (single-turn) GRPO with Q^π as the reward model.

Importantly, this lets us implement outcome-driven learning *entirely using standard single-turn RLHF components*, without bespoke multi-turn algorithms and implementations. In this work, we make the following contributions.

- **Multi-turn to single-turn RLHF reduction.** We give a formal reduction of multi-turn, response-level problem to a series of standard single-turn, token-level RLHF problems, enabling the direct use of off-the-shelf RLHF tooling. We prove a key insight: solving this single-turn RLHF problem with a standard token-level RLHF approach (e.g., GRPO or PPO) can be viewed as an approximate policy improvement step within the multi-turn problem.
- **Iterative GRPO.** We propose *Iterative GRPO*, a simple *batch online* method that alternates between fitting Q^π from trajectories and improving the policy π with standard GRPO on the learned Q^π (see Figure 1). Under the approximate policy improvement observation described above, Iterative GRPO can be interpreted as *approximate policy iteration* across successive online data-collection batches.
- **Empirical results.** Using six multi-turn environments, we illustrate that Iterative GRPO outperforms single-turn and multi-turn baselines in a realistic deployment setting with multiple online batches. In each batch, we deploy a policy and then collect a batch of data under that policy.

2 Multi-Turn RLHF with Outcomes

We study post-training for conversational agents in a *multi-turn* setting where the interaction alternates between environment outputs x_t and agent actions a_t:

$$x_1, a_1, x_2, a_2, \ldots$$

In the standard setting, we can think of x_t simply as the message sent by the user (or customer), while a_t is the message sent by the agent. However, by allowing the definition of x_t to be flexible, we can instantiate other variations of the problem where additional information besides the user's message may arrive. In general, x_t is a generic environment output that bundles all of the information emitted between an agent's consecutive actions.

We model this as an MDP $(\mathcal{S}, \mathcal{A}, P, P_0, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition kernel, P_0 is the initial state distribution, R is the reward function, and $\gamma \in [0, 1)$ is the discount factor.

State representation. The state representation is the full history up to the current turn,

$$s_t = (x_1, a_1, x_2, a_2, \ldots, x_t) \in \mathcal{S}, \tag{1}$$

with initial state $s_1 = x_1$ drawn from P_0. A stochastic policy $\pi_\theta(a \mid s)$ maps states to action distributions.

Transition dynamics and termination. After the agent takes action a_t in s_t, the environment produces the next output

$$x_{t+1} \sim P(\cdot \mid s_t, a_t). \tag{2}$$

The MDP includes an absorbing terminal state $\perp$. After the agent takes a_t and the environment produces x_{t+1}, the episode may terminate. Termination can depend on x_{t+1} (e.g., through a terminal flag indicating "no further messages" or an outcome indicating successful resolution of the conversation). Once in $\perp$, the process remains there.

Rewards and objective. The reward can be any function of the state, action, and next environment output; we write $r_t = R(s_t, a_t, x_{t+1})$. Given a stochastic policy $\pi(a \mid s)$ and an initial state s_1, the objective is to maximize the expected discounted return

$$V^\pi(s_1) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \;\middle|\; s_1 = s,\ a_t \sim \pi\right].$$

Under $s_1 \sim P_0$, the objective is $J(\pi) = \mathbb{E}_{s_1 \sim P_0}[V^\pi(s_1)]$.

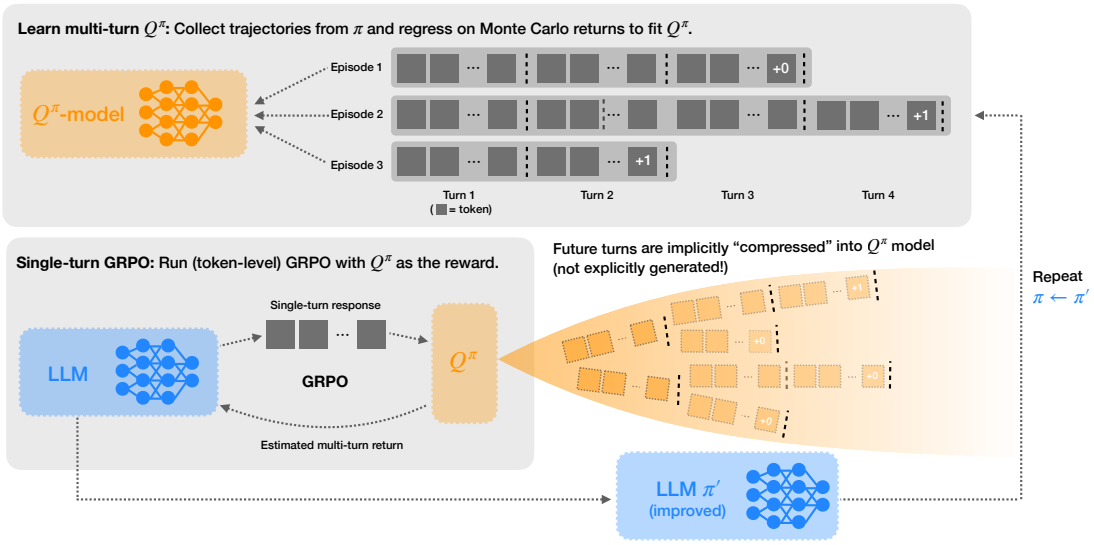

Figure 1: **Reducing multi-turn to single-turn RLHF via Iterative GRPO.** We visualize the main steps of our proposed approach. **Top:** First, we collect multi-turn trajectories under the current policy π, compute Monte Carlo returns, and use standard reward-modeling procedures to fit Q^π, a value function that predicts the expected total multi-turn reward under π. **Bottom:** Second, holding Q^π fixed, we run standard single-turn (token-level) GRPO using Q^π as the reward model. All future turns are thus implicitly "compressed" into Q^π, eliminating the need for any explicit handling of multi-turn trajectories. **Right:** Iterating this procedure ($\pi \leftarrow \pi'$) yields multi-turn improvements using only single-turn RLHF tools. Since we proceed in a series of online batches, we refer to this as "batch online."

Example 2.1 (Outcome-driven multi-turn RL)**.** *Here, we define the outcome-driven multi-turn conversational RL problem. Let the agent's message be a_t. Let c_t denote the customer message at turn t, and let o_t denote an* outcome *signal (e.g., task completion, issue resolution, or purchase) that is revealed after the customer's message c_t arrives. We set $x_t := (c_t, o_t)$, so that the interaction has the form $(c_1, o_1), a_1, (c_2, o_2), a_2, \ldots, (c_\tau, o_\tau)$. We suppose that termination occurs at the first outcome, i.e., the first time τ where $o_\tau = 1$. There is a sparse outcome reward $R(s_t, a_t, x_{t+1})$ that is only nonzero when $o_{t+1} = 1$.*

3 Multi-Turn to Single-Turn Reduction via Iterative GRPO

In the single-turn setting, actions are typically defined at the *token level*,[1] with planning performed over the sequence of tokens that constitute a response (Christiano et al., 2017; Ziegler et al., 2019). The state is the sequence of tokens generated so far, and the transition dynamics are to concatenate the latest action (token) with the previous state. In most instantiations, the overall response is then scored according to a reward model after the end of sequence (EOS) token.

In contrast, the *multi-turn* formulation naturally operates at the *response level*. This allows for conversation-level planning that occurs over the sequence of responses (Li et al., 2016; Wei et al., 2018; Shani et al., 2024). Unlike token-level single-turn RLHF, where the "dynamics" are just deterministic token concatenation, the multi-turn setting involves unknown environment dynamics P that govern how the conversation evolves. This places multi-turn RLHF closer to classical RL, and makes the learning problem substantially richer and more challenging.

In this work, we consider an approach inspired by dynamic programming where we reduce the initial multi-turn problem into a sequence of single-turn problems.

[1]There also exists response-level actions in a single-turn setting (Ahmadian et al., 2024), but in that case the problem is only a single stage, similar to a bandit problem.

3.1 THE REDUCTION

We now give two concrete observations that capture our *policy iteration* perspective on multi-turn RL. The following observation captures the essence of this reduction.

Observation 3.1 (GRPO is a policy improvement operator)**.** *Let ρ denote a distribution over single-turn states s (i.e., initial prompts). For a reward $R(s, a)$ over states s and completions a, define*

$$\texttt{SingleTurnOpt}(R) \in \arg\max_{\pi} \ \mathbb{E}_{s\sim\rho}\, \mathbb{E}_{a\sim\pi(\cdot|s)}\big[R(s,a)\big].$$

This is the standard single-turn RLHF problem. In practice, we approximately solve this objective with a token-level RLHF method such as GRPO.

Now suppose that Q^π, the state-action value function of a multi-turn policy π, is given to us. A policy that is greedy in the response-level action space with respect to Q^π improves upon π by the classical policy improvement theorem (Sutton et al., 1998). Our key insight is that since optimizing greedily in response space is difficult, we can instead run a single-turn RLHF optimizer with reward model Q^π, i.e.,

$$\pi' = \texttt{SingleTurnOpt}(Q^\pi). \tag{3}$$

If `SingleTurnOpt` *works well, then π' should be an improved version of π, in the sense that $V^{\pi'}(s) \geq V^\pi(s)$ at all states s. Moreover, if were given Q^*, then running* $\texttt{SingleTurnOpt}(Q^*)$ *gives us an approximation of π^*.*

Notice that in order to guarantee policy π' improves globally over π, we can relax the assumption that π' is greedy with respect to Q^π and only assume a local improvement property, as expressed in the following result (whose proof is given in Appendix B).

Theorem 3.2. *Assume we are able to obtain a (multi-turn) policy π' which produces a single-turn improvement over π from any state s, in the sense that*

$$\mathbb{E}_{a\sim\pi'(\cdot|s)}\left[Q^\pi(s,a)\right] \geq \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^\pi(s,a)\right].$$

Then the resulting (multi-turn) policy π' is globally better than π, in the sense that $V^{\pi'}(s_0) \geq V^\pi(s_0)$ for any initial state s_0.

The requirement that the new policy π' produces a single-turn improvement over π can be achieved by any RLHF technique that is effective in the setting of LLMs, such as GRPO (Shao et al., 2024), PPO (Schulman et al., 2017), or REINFORCE (Ahmadian et al., 2024). In this work, we use GRPO to implement the policy improvement step. Another version of Theorem 3.2, specialized to the case of KL-regularized objectives, is given in the Appendix.

Remark 3.3. *Observation 3.1 follows from the classical policy improvement theorem. Our key insight is that policy improvement provides a perspective for bridging token-level and response-level RL:* ***a token-level RL method can serve as a policy improvement operator for the response-level problem****. This is appealing because token-level RL methods are empirically effective in single-turn settings, and Observation 3.1 lets us directly leverage these mature methods and implementations.*

In Observation 3.1, we assumed that we have access to Q^π. In our next observation, we show that approximating Q^π can be achieved with standard supervised reward-model learning techniques used in single-turn RLHF.

Observation 3.4 (Reward model fitting as policy evaluation)**.** *Consider the standard single-turn RLHF setup. Let $\mathcal{D}_{\text{single}} = \{((s_i, a_i), r_i)\}_{i=1}^N$ be a supervised dataset of prompts s_i, responses a_i, and scalar targets r_i (e.g., human labels). Reward modeling with scalar targets is a supervised learning problem:*

$$\texttt{SingleTurnFitReward}(\mathcal{D}_{\text{single}}) \in \operatorname*{argmin}_{R} \ \mathbb{E}_{((s,a),r)\sim\mathcal{D}_{\text{single}}}\Big[\ell\big(R(s,a), r\big)\Big],$$

where R belongs to some function class (e.g., a transformer) and ℓ is a supervised loss (e.g., mean squared error).

Now consider a multi-turn policy π and a dataset of N on-policy trajectories $\{\tau^{(i)}\}_{i=1}^N$ collected by rolling out π. Each trajectory is of the form

$$\tau^{(i)} = \big(s_1^{(i)}, a_1^{(i)}, r_1^{(i)}, s_2^{(i)}, a_2^{(i)}, r_2^{(i)}, \ldots, s_{T_i}^{(i)}\big),$$

where T_i is the (possibly varying) horizon of trajectory i. For each time step $t \in \{1, \ldots, T_i - 1\}$, define the so-called Monte Carlo return to be

$$G_t^{(i)} \;=\; \sum_{k=0}^{T_i - t - 1} \gamma^k \, r_{t+k}^{(i)},$$

where $\gamma \in [0, 1)$ is the discount factor. We can then form a supervised dataset of state-action pairs and returns,

$$\mathcal{D}^{\pi}_{\text{multi}} = \left\{ \left((s_t^{(i)}, a_t^{(i)}), G_t^{(i)} \right) : i \in [N],\ t \in [T_i - 1] \right\}. \tag{4}$$

Now, it is straightforward to see that Monte Carlo policy evaluation Sutton et al. (1998) can be written as:

$$Q^{\pi} \approx \texttt{SingleTurnFitReward}(\mathcal{D}^{\pi}),$$

a single-turn RLHF operation.[2]

Remark 3.5. *In Observation 3.4, we made use of the Monte Carlo variant of policy evaluation in order to most easily leverage single-turn reward modeling tools. Other methods for performing policy evaluation include SARSA Sutton et al. (1998) and off-policy Q-evaluation (Sutton et al., 1998; Lagoudakis & Parr, 2003). These methods both use temporal differences (TD), which can have lower variance than Monte Carlo, but the downside is that TD-learning in an LLM setting is expensive and potentially unstable (Hong et al., 2024). In multi-turn settings with a reasonable horizon (i.e., the number of turns), Monte Carlo policy evaluation via standard reward modeling should suffice and has the added benefit of simplicity. However, if the data are collected using a policy other than π_i, then off-policy Q-evaluation would be required.*

Remark 3.6. *Multi-turn RL can be formulated as a monolithic token-level optimization problem (e.g., via PPO). However, this is often impractical. Since environment transitions depend on external user responses, a token-level trainer needs to "wait" for customer responses. In addition, token-level methods may suffer from credit assignment challenges, as sparse terminal rewards must be propagated through thousands of tokens across multiple turns. Our reduction avoids these pitfalls by using a Q-function to compress the long-horizon dependency into a response-level reward, enabling the use of stable single-turn RLHF tools.*

3.2 The Iterative GRPO Algorithm

We propose *Iterative GRPO*: at each round, we learn a Q-value function Q^{π} that estimates the (multi-turn) value of the current policy π at the response level, and then run a single-turn, token-level RLHF method (in our case GRPO) using Q^{π} as the reward model to obtain an improved policy π'. The justification for this algorithm comes directly from Observations 3.1 and 3.4, which together constitute a single step of *policy iteration*.

The pseudocode for Iterative GRPO is given in Algorithm 1. As we mentioned above, one of the main advantages of Iterative GRPO is that it can directly leverage mature and widely available off-the-shelf single-turn RLHF tools (von Werra et al., 2020; Lefaudeux et al., 2022; Sheng et al., 2024).

Iterative GRPO is a partially online approach, operating in a series of large batches (i.e., the collection of $\mathcal{D}^{\pi_i}_{\text{multi}}$). This type of *batch online* method is more stable and safer than continual online updates while being able to make larger improvements as compared to a fully offline method that is limited to optimizing within a close neighborhood of the behavioral policy.

3.3 Implementation via a Series of A/B Tests

In practice, π_0 consists of simply prompting a base LLM to generate relevant suggestions based on the conversation session and other contextual information. Trajectories from π_0 can be collected by

[2] Why do we focus on Q^{π} rather than aim for Q^*? This would likely involve an algorithm like fitted Q-iteration, but note that this requires the implementation of a max operator (over responses) to generate Q-labels. This could be another GRPO step, but the number of such iterations is inherently limited, since repeated policy improvement steps can quickly push the policy into regions of the response space that are out-of-distribution with respect to the original data. This is why we leverage the ability to do periodic data collection and perform Q-evaluation for the response-level problem.

Algorithm 1 Iterative GRPO for Multi-Turn RL

Initialize π_0 (e.g. prompt an instruction-tuned model to suggest a response given conversation). Set number of online batches K.
Initialize $i \leftarrow 0$.
while $i < K$ **do**
  Deploy π_i and collect trajectories $\{\tau^{(i)}\}_{i=1}^{N}$.
  Create the dataset $\mathcal{D}_{\text{multi}}^{\pi_i}$ according to equation 4.
  Compute $Q^i \leftarrow \texttt{SingleTurnFitReward}(\mathcal{D}_{\text{multi}}^{\pi_i})$.
  Compute $\pi_{i+1} \leftarrow \texttt{SingleTurnOpt}(Q^i)$.
  Update $i \leftarrow i + 1$.
end while
return π_K.

deploying it through an A/B test, from which we can learn an approximation to Q^{π_0} (see top part of Figure 1). After a policy improvement step (i.e., running GRPO on Q^{π_0}; see bottom part of Figure 1) that results in a new policy π_1, we can deploy π_1 in an A/B test and track outcome metrics, such as the number of purchases. This A/B test for π_1 also serves to collect trajectories, from which we can repeat the above steps to arrive at π_2, and so on. Once we are satisfied with a particular policy π_K, it can be deployed to production.

3.4 Action Coverage and Off-Policy Data Collection

One important consideration is the coverage of the action space of each iteration's collected dataset, which affects how well the learned Q-function generalizes across the action space. This can be an issue if, for example, $\pi_0(\cdot \mid s)$ is near deterministic and we use on-policy Monte Carlo policy evaluation. The resulting $Q^{\pi_0}(s, a)$ would therefore only be reliable at a small set of actions a that are covered by $\pi_0(\cdot \mid s)$. To resolve this, we may need to consider collecting data using a behavioral policy π_0' that is significantly different from π_0 in the sense that it is much more exploratory in its actions. For example, one way to do this is via *persona exploration* where we can prompt the LLM to randomly select an array of characteristics such as tone, emoji usage, response length, etc. The trade-off with collecting data using π_0' is that we must now perform an off-policy evaluation to learn Q^{π_0} from trajectories collected from π_0'. However, this becomes more complex than on-policy Monte Carlo policy evaluation since it involves a TD-based procedure (Lagoudakis & Parr, 2003).

4 Numerical Experiments

4.1 Multi-Turn Environments

We evaluate our method across six multi-turn negotiation environments. In each environment, two agents interact in natural language over multiple turns and receive a scalar reward at the end of the dialogue. These environments differ along four main dimensions: (1) the number of issues under negotiation, ranging from single-issue price bargaining to five-issue negotiations over multiple attributes; (2) the extent to which the setting permits joint gains, ranging from purely competitive settings to ones where differences in preferences allow mutually beneficial agreements; (3) the information available to each agent, ranging from fully shared information to private valuations, priorities, and utility functions; and (4) the size and structure of the agreement space, ranging from a single continuous decision to combinatorial outcome spaces with thousands of possible agreements.

Unless otherwise noted, each environment consists of two LLM-based agents: (1) the trained agent π and (2) a static environment agent π_{env}. Both agents generate free-form natural language; there are no heuristics or scripted responses. Each agent receives a system prompt describing their role and private information (e.g., valuations, priorities, or utility functions). The agents then alternate turns of free-form natural language until a terminal event is reached, either a successful agreement or a breakdown. Both agents are simulated with *Qwen-2.5-7B-Instruct*. We give brief descriptions of the multi-turn environments below.

Craigslist Negotiation. This is a price negotiation task using the CraigslistBargain dataset (He et al., 2018), which contains negotiation dialogues collected from actual Craigslist transactions across six product categories. The trained agent π plays a vendor negotiating on behalf of sellers, aiming to make sales at favorable prices, while π_{env} plays a customer seeking a good price. The vendor sees the listing price; the customer sees the listing price and a private target price. The reward for π is the ratio of the final sale price to the listing price, or zero if no sale is made. Full prompt details are given in Appendix E.

Ultimatum Game. In the Ultimatum Game (Bianchi et al., 2024), two agents negotiate how to split a money pool, with scenarios procedurally generated from a cross-product of pool sizes and round counts. π plays a proposer who holds the entire money pool and offers a split, while π_{env} plays a responder who can accept, reject, or counter-propose. Both players observe the pool size and the number of negotiation rounds; neither has private information. The reward for π is its share divided by half the pool size (equal split yields 1.0; no deal yields zero, as both players receive nothing). Full prompt details are given in Appendix F.

Deal or No Deal. In the Deal or No Deal task (Lewis et al., 2017), two agents must divide a shared pool of items (books, hats, and balls) with varying counts. π plays one negotiator aiming to claim items that maximize its private point total, while π_{env} plays the opposing negotiator with the same goal. Each agent sees the item counts and its own private value for each item type but not the other agent's values. Each agent's total value across all items is 10 points, though both agents cannot simultaneously achieve this maximum. The reward for π is the points earned divided by 5 (the score from an equal split), or zero for no deal. Full prompt details are given in Appendix G.

CaSiNo. In the CaSiNo task (Chawla et al., 2021), two campers must divide 3 packages each of Food, Water, and Firewood (9 packages total), where each camper has a different private priority ordering (High/Medium/Low) over the three supply types. π plays one camper trying to maximize its point total, while π_{env} plays the opposing camper with a different priority ordering. Each camper sees only its own priority ranking; the other camper's priorities are unknown. Each package is worth 5, 4, or 3 points depending on whether the supply is High, Medium, or Low priority for that camper. The reward for π is the points earned divided by 18 (the score from an equal split), or zero for no deal. Full prompt details are given in Appendix H.

Job Interview. Adapted from the multi-issue negotiation task of Yamaguchi et al. (2021), this environment models a job offer negotiation with cross-issue dependencies and procedurally generated utility functions. π plays a recruiter negotiating the terms of a job offer, aiming to maximize its private utility. π_{env} plays a worker seeking favorable employment terms. The two parties negotiate over five issues (Salary, Weekly Holiday, Workplace, Company, and Position), where the value of Position depends on the chosen Company (a cross-issue dependency). Each party sees its own private utility function: a set of importance weights over issues and valuations over options, but cannot observe the other party's preferences. The reward for π is a weighted sum of normalized per-issue scores, yielding a value in $[0, 1]$ representing the fraction of maximum possible recruiter utility achieved (zero for no deal). Full prompt details are given in Appendix I.

Alliance Negotiation. Adapted from the LLM-Deliberation framework (Abdelnabi et al., 2024), two stakeholders negotiate over five project issues (Infrastructure, Environment, Employment, Funding, and Compensation), each with multiple possible options. π plays one stakeholder seeking a favorable agreement, while π_{env} plays the opposing stakeholder with opposing preferences (one stakeholder's preferred option on each issue tends to be the other's least preferred). Each stakeholder sees a private score matrix assigning point values to every option on every issue, as well as a private BATNA (Best Alternative to Negotiated Agreement) threshold representing the minimum acceptable total score. The reward for π is its deal score divided by 50 (half the maximum score of 100) when the agreed deal meets or exceeds its BATNA threshold, and zero for any deal below the threshold or for no deal. Full prompt details are given in Appendix J.

4.2 Experimental Setup

We simulate a realistic deployment setting in which a model is iteratively retrained and deployed over multiple rounds. In each round, the current policy is deployed to interact with the environment and collect a batch of trajectories; we then use this newly collected data to fine-tune the model before deploying it again. Each round generates 400 conversations (100 distinct scenarios, each simulated

4 times independently) over $K = 4$ rounds total. The per-turn discount factor is $\gamma = 0.9$, which encourages the agent to reach favorable outcomes efficiently. For each environment, we report the expected discounted return $J(\pi)$. Since our environments only produce a terminal reward at the end of the conversation, the total reward of an episode of length τ is simply $\gamma^{\tau-1} r_\tau$. This captures both the quality of the negotiated outcome and the efficiency with which it is reached. Results are averaged over 3 seeds. All algorithms are evaluated under the same iterative deployment protocol. We compare the following:

- **I-GRPO**: Our proposed approach (Algorithm 1), which applies multi-turn GRPO iteratively across deployment rounds.
- **ST-GRPO**: A myopic baseline that applies GRPO to each individual turn, treating them as independent decision problems ($\gamma = 0$).
- **MT-SFT**: A supervised baseline that selects high-reward rollouts and fine-tunes the model to imitate the assistant messages from these trajectories.
- **ArCHer** (Zhou et al., 2024): An off-policy actor-critic method for multi-turn RL with language models that learns a response-level Q-function via temporal difference learning. The original implementation uses a RoBERTa-based embedding model (Liu et al., 2019) as the critic.
- **ArCHer-LC**: An adaptation of ArCHer designed for longer-form, multi-turn conversations. ArCHer-LC (LLM-Critic) replaces the embedding model critic with a generative model critic that shares the same language model backbone as the policy, enabling the critic to better handle the longer contexts that arise in extended (implementation details in Appendix K).

Full experimental details are provided in Appendix L.

4.3 RESULTS

Table 1 and Figure 2 summarize the expected discounted return $J(\pi)$ across all six environments after $K = 4$ rounds of iterative training. I-GRPO achieves the highest final return in all six environments, improving over the base policy in every case. The gains are particularly large in Alliance Negotiation ($0.85 \rightarrow 1.23$) and Ultimatum ($0.77 \rightarrow 0.98$), where the base policy leaves substantial room for improvement. ST-GRPO, which treats each turn in isolation (equivalent to $\gamma = 0$), improves over the base policy in some environments but consistently underperforms I-GRPO, confirming the value of multi-turn credit assignment. MT-SFT degrades in most environments, suggesting that imitating high-reward trajectories without explicit RL optimization is insufficient.

Table 1: Expected return $J(\pi)$. "Base" reports Round 0; trained methods report Round 4.[3]

Method	Craigslist	Ultimatum	DealOrNoDeal	CaSiNo	Job Interview	Alliance
Base	0.40	0.77	1.08	0.84	0.26	0.85
MT-SFT	$0.37_{\pm 0.00}$	$0.74_{\pm 0.00}$	$1.01_{\pm 0.00}$	$0.79_{\pm 0.01}$	$0.21_{\pm 0.00}$	$0.70_{\pm 0.02}$
ArCHer	$0.43_{\pm 0.13}$	$0.82_{\pm 0.06}$	$0.74_{\pm 0.02}$	$0.66_{\pm 0.04}$	$0.09_{\pm 0.03}$	$0.53_{\pm 0.11}$
ArCHer-LC	$0.29_{\pm 0.13}$	$0.82_{\pm 0.03}$	$0.76_{\pm 0.13}$	$0.73_{\pm 0.09}$	$0.12_{\pm 0.00}$	$0.80_{\pm 0.33}$
ST-GRPO	$0.36_{\pm 0.05}$	$0.93_{\pm 0.03}$	$0.97_{\pm 0.02}$	$0.84_{\pm 0.02}$	$0.29_{\pm 0.04}$	$0.54_{\pm 0.21}$
I-GRPO	$\mathbf{0.48}_{\pm \mathbf{0.07}}$	$\mathbf{0.98}_{\pm \mathbf{0.00}}$	$\mathbf{1.17}_{\pm \mathbf{0.04}}$	$\mathbf{0.89}_{\pm \mathbf{0.03}}$	$\mathbf{0.36}_{\pm \mathbf{0.03}}$	$\mathbf{1.23}_{\pm \mathbf{0.06}}$

Both ArCHer variants struggle relative to I-GRPO. The original ArCHer degrades in most environments, consistent with the instability and critic truncation issues discussed in Appendix K. ArCHer-LC shows less degradation than the original, but still struggles to learn on the long-form multi-turn negotiation tasks tested, underperforming I-GRPO across all six environments.

Beyond the discounted return, I-GRPO training produces two notable effects. First, trained policies consistently shorten conversations while maintaining or improving agreement rates (Appendix D), reflecting the discount factor's incentive for efficient negotiation. Second, undiscounted terminal rewards also improve (Appendix D), confirming that agents learn to negotiate genuinely better outcomes, not merely faster ones.

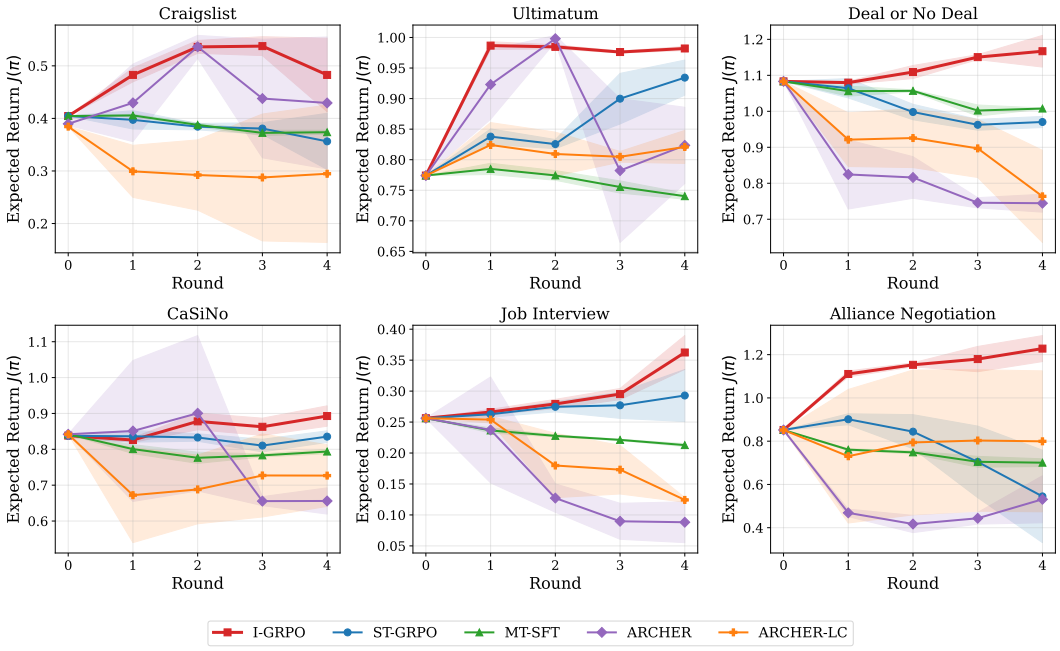

Figure 2: Expected discounted return $J(\pi)$ across six environments over four training rounds. Error bands show ± 1 SE across 3 seeds.

5 Related Work

Our work is most closely aligned with prior works that use RL to optimize multi-turn conversation trajectories rather than single responses; that is, settings where multiple future responses are considered at each step.

5.1 Multi-Turn Preference Optimization

Recent work on multi-turn preference optimization centers on *trajectory-level* preference feedback, where preferences are expressed over entire conversations. The MTPO algorithm of Shani et al. (2024) learns from comparisons over whole conversations and is designed for online preference collection using mirror-descent algorithm. Similar work includes Xiong et al. (2024) and Shi et al. (2024), both of which derive extensions of DPO (Rafailov et al., 2023) to the multi-turn setting. Xiong et al. (2024) proposes an iterative approach, while Shi et al. (2024) focuses on the offline setting. Our approach differs from these in a few key aspects. First, in the context of e-commerce conversational AI, collecting pairwise preference data of multi-turn trajectories is often impractical due to the length and complexity of conversations. Second, while prior works develop bespoke algorithms for multi-turn optimization, our method instead provides a simple reduction to established single-turn techniques. Lastly, although Shani et al. (2024) and Shi et al. (2024) focus on fully online and offline regimes respectively, our approach is closer to Xiong et al. (2024), operating in a hybrid setting. This enables us to retain the adaptability of online learning while mitigating the safety and stability concerns associated with continual online updates.

5.2 Hierarchical Multi-Turn RL

Perhaps the most closely related work is Zhou et al. (2024). Like our approach, the ArCHer method distinguishes between response-level and token-level problems. However, the underlying methods differ. ArCHer presents a hierarchical framework where a Q-function is fitted using off-policy Bellman updates that runs in parallel to token-level optimization using the learned Q-function as the reward model. Critically, our approach relies on the insight that the token-level optimization is in fact a policy improvement step in disguise (Thms. 3.2 and C.1). ArCHer uses methods like Implicit Q-Learning (IQL) Kostrikov et al. (2022) as the off-policy Bellman update since it does not require

a direct implementation of the max operator. IQL uses an upper expectile to estimate max Q-values and it is not guaranteed to learn an optimal state-action value function.

In contrast, our reduction mechanism demonstrates that token-level optimization is a policy improvement step, which in turn motivates our policy iteration-style algorithm. This insight, at least in principle, guarantees that our algorithm will eventually converge to an optimal policy (with enough iterations) whereas ArCHer provides no such guarantees. Another important difference lies in the simplicity of our method. ArCHer employs TD learning and optimizes multiple critic networks (Q, V, and baseline networks), which often necessitates additional tricks and extensive hyperparameter tuning to ensure stability. In contrast, our algorithm can be implemented in two straightforward phases: a supervised Q-evaluation step, followed by GRPO.

POAD (Wen et al., 2024) is an orthogonal approach that extends PPO by decomposing actions into groups of tokens rather than individual tokens. REFUEL (Gao et al., 2025) simplifies ArCHer by replacing the two-step critic and policy optimization with a regression procedure. However, similar to preference optimization, REFUEL requires pairwise trajectory rollouts for each state, which can be impractical to simulate in a real-world e-commerce setting. Recently, in the context of AI tutoring, Nam et al. (2025) proposed a method that transforms the dialogue history into a compact numerical representation of the student's state and then defines an abstract MDP over four "high-level" actions. This can be viewed as another type of hierarchical approach since the high-level action is then used to condition an LLM's responses to the student.

5.3 Other Offline Approaches

Several prior works have focused on optimizing dialogue using offline methods applied to static datasets (Jaques et al., 2020; Jang et al., 2022; Snell et al., 2022; Verma et al., 2022; Chen et al., 2025). Notably, Chen et al. (2025) performs inference-time search on a model trained offline. In contrast, our approach is designed to support periodic online updates. This periodic updating not only distinguishes our method from fully offline approaches, but also enables a simpler algorithmic framework based on approximate policy iteration.

6 Concluding Remarks

We show multi-turn RL can be reduced to a series of single-turn RLHF-style problems, motivating our online batch policy iteration algorithm, Iterative GRPO. Empirically, we introduce a novel multi-turn environment, the Craigslist Negotiation Environment, where we show that I-GRPO improves upon multi-turn and single-turn baselines.

REFERENCES

Sahar Abdelnabi, Amr Gomaa, Sarath Sivaprasad, Lea Schönherr, and Mario Fritz. Cooperation, competition, and maliciousness: Llm-stakeholders interactive negotiation. (arXiv:2309.17234), 2024. doi: 10.48550/arXiv.2309.17234. URL http://arxiv.org/abs/2309.17234. arXiv:2309.17234.

Alekh Agarwal, Nan Jiang, Sham M Kakade, and Wen Sun. Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep*, 32:96, 2019.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in LLMs. *arXiv preprint arXiv:2402.14740*, 2024.

Federico Bianchi, Patrick John Chia, Mert Yuksekgonul, Jacopo Tagliabue, Dan Jurafsky, and James Zou. How well can llms negotiate? negotiationarena platform and analysis. (arXiv:2402.05863), February 2024. doi: 10.48550/arXiv.2402.05863. URL http://arxiv.org/abs/2402.05863. arXiv:2402.05863.

Kushal Chawla, Jaysa Ramirez, Rene Clever, Gale Lucas, Jonathan May, and Jonathan Gratch. Casino: A corpus of campsite negotiation dialogues for automatic negotiation systems. (arXiv:2103.15721), April 2021. doi: 10.48550/arXiv.2103.15721. URL http://arxiv.org/abs/2103.15721. arXiv:2103.15721.

Zhiliang Chen, Xinyuan Niu, Chuan-Sheng Foo, and Bryan Kian Hsiang Low. Broaden your SCOPE! efficient multi-turn conversation planning for llms with semantic space. *arXiv preprint arXiv:2503.11586*, 2025.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Zhaolin Gao, Wenhao Zhan, Jonathan D. Chang, Gokul Swamy, Kianté Brantley, Jason D. Lee, and Wen Sun. Regressing the relative future: Efficient policy optimization for multi-turn RLHF. *arXiv preprint arXiv:2410.04612*, 2025.

He He, Derek Chen, Anusha Balakrishnan, and Percy Liang. Decoupling strategy and generation in negotiation dialogues, 2018.

Joey Hong, Anca Dragan, and Sergey Levine. Q-SFT: Q-learning for language models via supervised fine-tuning. *arXiv preprint arXiv:2411.05193*, 2024.

Youngsoo Jang, Jongmin Lee, and Kee-Eung Kim. Gpt-critic: Offline reinforcement learning for end-to-end task-oriented dialogue systems. In *International Conference on Learning Representations*, 2022.

Natasha Jaques, Judy Hanwen Shen, Asma Ghandeharioun, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Shane Gu, and Rosalind Picard. Human-centric dialog training via offline reinforcement learning. *arXiv preprint arXiv:2010.05848*, 2020.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. In *International Conference on Learning Representations*, 2022.

Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.

Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, Daniel Haziza, Luca Wehrstedt, Jeremy Reizenstein, and Grigory Sizov. xformers: A modular and hackable transformer modelling library. https://github.com/facebookresearch/xformers, 2022.

Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, and Dhruv Batra. Deal or no deal? end-to-end learning for negotiation dialogues. (arXiv:1706.05125), 2017. doi: 10.48550/arXiv.1706.05125. URL http://arxiv.org/abs/1706.05125. arXiv:1706.05125.

Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 1192–1202, 2016.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach, July 2019. URL http://arxiv.org/abs/1907.11692. arXiv:1907.11692.

Hyunji Nam, Omer Gottesman, Amy Zhang, Dean Foster, Emma Brunskill, and Lyle Ungar. Efficient RL for optimizing conversation level outcomes with an LLM-based tutor. *arXiv preprint arXiv:2507.16252*, 2025.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Lior Shani, Aviv Rosenberg, Asaf Cassel, Oran Lang, Daniele Calandriello, Avital Zipori, Hila Noga, Orgad Keller, Bilal Piot, Idan Szpektor, et al. Multi-turn reinforcement learning with preference human feedback. *Advances in Neural Information Processing Systems*, 37:118953–118993, 2024.

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Wentao Shi, Mengqi Yuan, Junkang Wu, Qifan Wang, and Fuli Feng. Direct multi-turn preference optimization for language agents. *arXiv preprint arXiv:2406.14868*, 2024.

Charlie Snell, Ilya Kostrikov, Yi Su, Mengjiao Yang, and Sergey Levine. Offline RL for natural language generation with implicit language Q learning. *arXiv preprint arXiv:2206.11871*, 2022.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Siddharth Verma, Justin Fu, Mengjiao Yang, and Sergey Levine. Chai: A chatbot AI for task-oriented dialogue with offline reinforcement learning. *arXiv preprint arXiv:2204.08426*, 2022.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformer reinforcement learning. https://github.com/huggingface/trl, 2020.

Wei Wei, Quoc Le, Andrew Dai, and Jia Li. Airdialogue: An environment for goal-oriented dialogue research. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3844–3854, 2018.

Muning Wen, Ziyu Wan, Weinan Zhang, Jun Wang, and Ying Wen. Reinforcing language agents via policy optimization with action decomposition. *arXiv preprint arXiv:2405.15821*, 2024.

Wei Xiong, Chengshuai Shi, Jiaming Shen, Aviv Rosenberg, Zhen Qin, Daniele Calandriello, Misha Khalman, Rishabh Joshi, Bilal Piot, Mohammad Saleh, et al. Building math agents with multi-turn iterative preference learning. *arXiv preprint arXiv:2409.02392*, 2024.

Atsuki Yamaguchi, Kosui Iwasa, and Katsuhide Fujita. Dialogue act-based breakdown detection in negotiation dialogues. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (eds.), *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 745–757, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.63. URL https://aclanthology.org/2021.eacl-main.63/.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn RL. *arXiv preprint arXiv:2402.19446*, 2024.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A LLM USAGE

The authors used LLMs to polish and improve the clarity of the writing in this paper.

B PROOF OF THEOREM 3.2

Proof. By the performance difference lemma (see, e.g., Section 1.5 of Agarwal et al. (2019)), we have

$$\begin{aligned} V^{\pi'}(s_0) - V^{\pi}(s_0) &= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi'}_{s_0}}\,\mathbb{E}_{a\sim\pi'(\cdot|s)}\big[A^{\pi}(s,a)\big] \\ &= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi'}_{s_0}}\,\Big[\mathbb{E}_{a\sim\pi'(\cdot|s)}\big[Q^{\pi}(s,a)\big] - \mathbb{E}_{a\sim\pi(\cdot|s)}\big[Q^{\pi}(s,a)\big]\Big] \geq 0, \end{aligned}$$

where

$$d^{\pi'}_{s_0}(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t\,\mathbb{P}(s_t = s \mid s_0, \pi') \tag{5}$$

is the discounted state visitation distribution under π'. □

C APPROXIMATE KL-REGULARIZED POLICY IMPROVEMENT

We now specialize Theorem 3.2 to the case of an "idealized" version of PPO (Schulman et al., 2017); namely, we include the KL-regularization term in the policy improvement objective (this is similar to the PPO objective without the clipping term)[4]. Here, we also allow for Q^{π} to be estimated with some maximum norm error ϵ_Q and allow the optimization objective to be optimized with some slack δ_s for each state s.

Theorem C.1. *Consider a multi-turn policy π. Suppose we are given $\widehat{Q}$, an approximation to Q^{π} with max-norm error $\|\widehat{Q} - Q^{\pi}\|_{\infty} \leq \epsilon_Q$. Suppose we construct a new policy π' by maximizing the KL-regularized objective $\mathcal{L}(\pi') = \mathbb{E}_{s\sim P_0}\,\mathcal{L}_s(\pi')$, where*

$$\mathcal{L}_s(\pi') = \mathbb{E}_{a\sim\pi'(\cdot\,|\,s)}\big[\widehat{Q}(s,a)\big] - \beta\,\mathrm{KL}\big(\pi(\cdot\,|\,s)\,\|\,\pi'(\cdot\,|\,s)\big),$$

for some initial state distribution P_0 and a constant $\beta > 0$. Assume that for each s, our optimization achieves $\mathcal{L}_s(\pi') \geq \sup_{\mu}\mathcal{L}_s(\mu) - \delta_s$ for some error δ_s. Then, π' is an improved policy compared to π if ϵ_Q and δ_s are small enough.

Proof. By feasibility of π, we have:

$$\begin{aligned} \mathcal{L}_s(\pi') &\geq \sup_{\mu}\mathcal{L}_s(\mu) - \delta_s \\ &\geq \mathcal{L}_s(\pi) - \delta_s \\ &= \mathbb{E}_{a\sim\pi(\cdot\,|\,s)}\big[\widehat{Q}(s,a)\big] - \delta_s \end{aligned} \tag{6}$$

where the final equality follows by the fact that the KL penalty term is 0 in $\mathcal{L}_s(\pi)$. Adding the term $\mathbb{E}_{a\sim\pi'(\cdot\,|\,s)}\big[Q^{\pi}(s,a)\big] - \mathbb{E}_{a\sim\pi(\cdot\,|\,s)}\big[Q^{\pi}(s,a)\big]$ to both sides of Equation 6 and rearranging, we get

$$\mathbb{E}_{a\sim\pi'(\cdot\,|\,s)}\big[Q^{\pi}(s,a)\big] - \mathbb{E}_{a\sim\pi(\cdot\,|\,s)}\big[Q^{\pi}(s,a)\big] \geq \beta\,\mathrm{KL}\big(\pi(\cdot\,|\,s)\,\|\,\pi'(\cdot\,|\,s)\big) - \delta_s - 2\epsilon_Q \tag{7}$$

where we use the max-norm error bound which implies that $|\hat{Q}(s,a) - Q^{\pi}(s,a)| \leq \epsilon_Q$ for all (s,a). Since $\mathbb{E}_{a\sim\pi(\cdot\,|\,s)}\big[Q^{\pi}(s,a)\big] = V^{\pi}(s)$, Equation 7 implies

$$\mathbb{E}_{a\sim\pi'(\cdot\,|\,s)}\big[A^{\pi}(s,a)\big] \geq \beta\,\mathrm{KL}\big(\pi(\cdot\,|\,s)\,\|\,\pi'(\cdot\,|\,s)\big) - \delta_s - 2\epsilon_Q. \tag{8}$$

[4]Note that the policy improvement objective can be equivalently written as the "surrogate advantage objective" as introduced in TRPO (Schulman et al., 2015). To see this, $\mathbb{E}_{a\sim\pi'(\cdot\,|\,s)}\big[\widehat{Q}(s,a)\big] = \mathbb{E}_{a\sim\pi(\cdot\,|\,s)}\big[\frac{\pi'(a|s)}{\pi(a|s)}\widehat{Q}(s,a)\big]$.

By the performance difference lemma (see, e.g., Section 1.5 of Agarwal et al. (2019)), we have

$$\begin{aligned} V^{\pi'}(s_0) - V^{\pi}(s_0) &= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_{s_0}^{\pi'}}\,\mathbb{E}_{a\sim\pi'(\cdot|s)}\big[A^{\pi}(s,a)\big] \\ &\geq \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_{s_0}^{\pi'}}\left[\beta\,\mathrm{KL}\big(\pi(\cdot\,|\,s)\,\|\,\pi'(\cdot\,|\,s)\big) - \delta_s - 2\epsilon_Q\right], \end{aligned}$$

where $d_{s_0}^{\pi'}$ is as defined in Equation equation 5. Since the KL term is non-negative, for small enough δ_s and ϵ_Q, the policy π' obtained by optimizing the KL-regularized policy improvement objective is an improved policy. □

D Additional Metrics

D.1 Conversation Length

Figure 3 shows the average conversation length (number of turns) across environments over training rounds. I-GRPO consistently reduces conversation length while maintaining or improving agreement rates, reflecting the discount factor's incentive for efficient negotiation. By Round 4, most environments converge to near-minimal conversation lengths (3–5 turns).

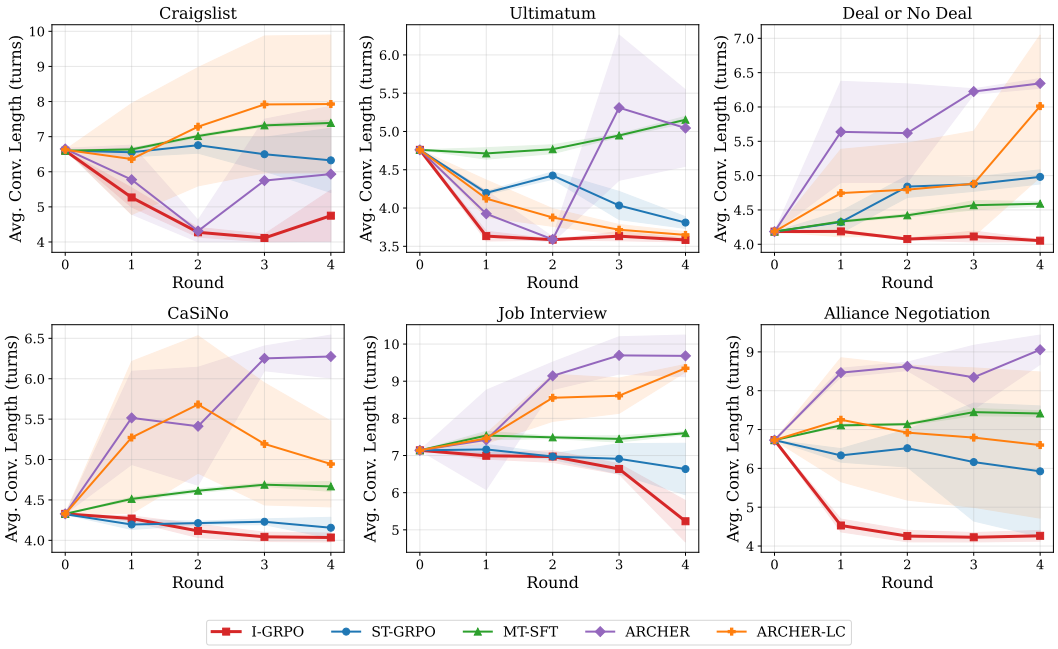

Figure 3: Average conversation length across six environments over four training rounds. Error bands show ±1 SE across 3 seeds.

D.2 TERMINAL REWARD (UNDISCOUNTED)

Figure 4 shows the undiscounted terminal reward r^τ across environments. Since the discounted return $\gamma^{\tau-1} r^\tau$ can improve either by shortening conversations or by achieving better outcomes, tracking r^τ in isolation confirms that I-GRPO improves outcome quality independently of efficiency gains. Terminal reward increases in all six environments, with the largest gains in Alliance Negotiation and Ultimatum.

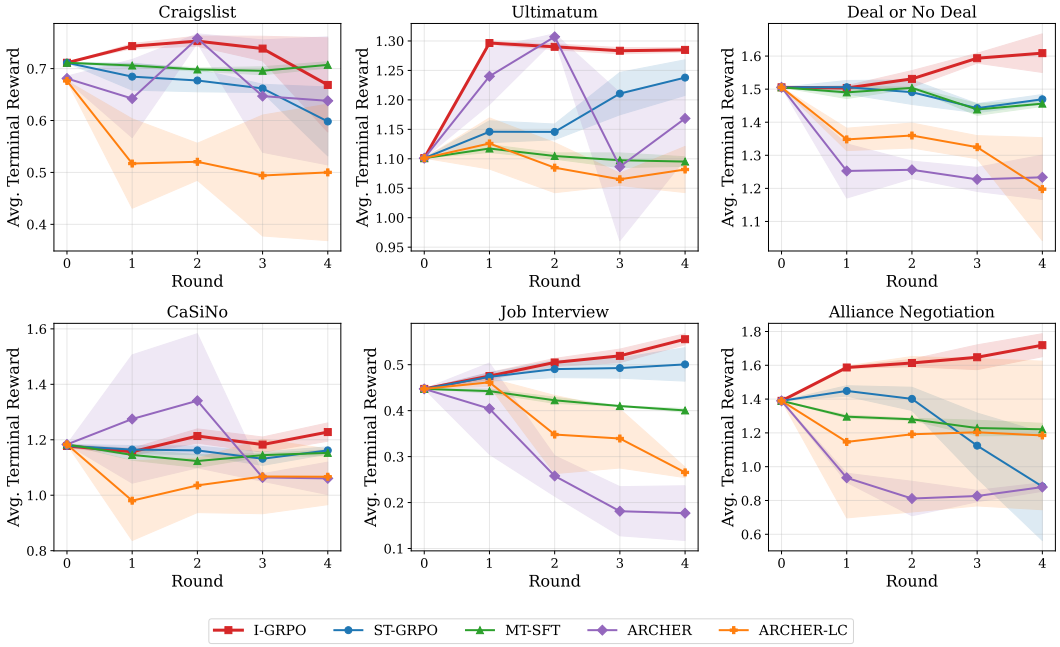

Figure 4: Undiscounted terminal reward $R(o)$ across six environments over four training rounds. Error bands show ± 1 SE across 3 seeds.

E CRAIGSLIST NEGOTIATION ENVIRONMENT DETAILS

E.1 AGENT PROMPTS

Both agents receive minimal system prompts that establish their role and pricing information. The vendor prompt specifies only the item description and listing price; the agent must learn its negotiation strategy entirely through RL. The customer prompt specifies the item description, listing price, and a target price derived from the original human negotiations. Each prompt includes rules instructing the agent to signal agreement with `\boxed{AGREED: $X}` or walk away with `\boxed{NO DEAL}`. The full prompts are provided below.

Vendor System Prompt

```
You are selling: [item description]

Your listing price is $[listing price]. Negotiate to get the best
possible price.

RULES:
- When you reach an agreement, respond with exactly: \boxed{AGREED:
$<price>}
- If no agreement can be reached, respond with: \boxed{NO DEAL}
```

Customer System Prompt

```
You are buying: [item description]

The listing price is $[listing price]. Your target price is $[buyer
target].

RULES:
- Negotiate to get as close to your target price of $[buyer target] as
possible
- When you reach an agreement, respond with exactly: \boxed{AGREED:
$<price>}
- If the price is too high, respond with: \boxed{NO DEAL}
```

The vendor starts each negotiation with a scripted opening:

Vendor Opening Message

```
Hello! I'm selling [item description].

My asking price is $[listing price]. Are you interested?
```

E.2 DATA SOURCE AND ECONOMIC PARAMETERS

The CraigslistBargain dataset (He et al., 2018) provides real listing information (item titles, descriptions, listing prices) and buyer target prices from the original human negotiations across six product categories: housing, furniture, cars, bikes, phones, and electronics. Each negotiation scenario uses two parameters from the dataset:

- **Listing price** (p_{list}): The vendor's asking price, taken directly from the Craigslist listing.
- **Buyer target** (p_{target}): The buyer's target price, from the original human negotiation data.

The vendor sees only the listing price; the customer sees both the listing price and their target. No additional economic parameters (reservation prices, cost bases, budgets) are provided to either agent. The vendor must discover its negotiation strategy entirely through reinforcement learning.

E.3 Reward Structure

We use a *price ratio* reward that measures the agreed price relative to the listing price:

$$R(s_\tau, a_\tau, x_{\tau+1}) = \begin{cases} \dfrac{p}{p_{\text{list}}} & \text{if agreement at price } p \\ 0 & \text{if no agreement} \end{cases}$$

This reward encourages the vendor to close deals while also incentivizing higher prices. Selling at the listing price yields a reward of 1.0; selling below yields a reward strictly less than 1.0. For example, if the listing price is \$40 and the deal closes at \$32, the reward is $32/40 = 0.8$. If no deal is reached, the reward is 0.

E.4 Before and After Training

The following shows the same scenario (Ballard Designs rug, listed at \$100, buyer target \$60) before and after I-GRPO training with $\gamma = 0.9$.

Before training (Round 0, 11 turns). The base model negotiates through repeated rounds of splitting the difference:

> **Vendor:** "My asking price is \$100.00. Are you interested?"
> **Customer:** "I'd like to offer \$60."
> **Vendor:** "I can do \$80."
> **Customer:** "How about \$70?"
> **Vendor:** "I can come down to \$75."
> **Customer:** "I'll go up to \$65."
> **Vendor:** "Let's meet at \$72."
> **Customer:** "\$68 is my final offer."
> $\cdots$ (continues for 11 total turns)
> **Outcome:** Agreement at \$68. Reward: $68/100 = 0.68$.

After training (Round 3, 4 turns). The trained vendor closes the deal immediately:

> **Vendor:** "My asking price is \$100.00. Are you interested?"
> **Customer:** "I'd like to offer \$60."
> **Vendor:** "I can accept \$60. \boxed{AGREED: \$60}"
> **Outcome:** Agreement at \$60. Reward: $60/100 = 0.60$.

The trained agent accepts a lower price but achieves a higher expected discounted return: $0.60 \cdot 0.9^4 = 0.39$ versus $0.68 \cdot 0.9^{11} = 0.21$. Under our reward formulation, the agent learns to close deals earlier rather than engage in extended back-and-forth negotiation.

E.5 Two Example Scenarios

Tracking a single scenario (iPhone 6 128GB, listed at \$320, buyer target \$288) across training rounds shows improvement on both dimensions. Conversations shorten from 8 to 4 turns while the agreed price increases from \$290 to \$300.

Round	Turns	Price	Sale Ratio
0 (base)	8	\$290	90.6%
1	6	\$295	92.2%
2	4	\$300	93.8%
3	4	\$300	93.8%

Not all scenarios improve monotonically. For a motorized bike engine kit (listed at \$150, buyer target \$82), the vendor initially improves after one round of training, closing in 4 turns at \$120 instead of 11 turns at \$109. By round 3, however, the vendor accepts the buyer's exact target price of \$82. The sale ratio drops to 54.7%, lower than the base model's 72.7%, despite the shorter conversation.

Round	Turns	Price	Sale Ratio
0 (base)	11	$109	72.7%
1	4	$120	80.0%
2	4	$120	80.0%
3	4	$82	54.7%

F ULTIMATUM GAME DETAILS

F.1 AGENT PROMPTS

The Ultimatum Game involves two agents: a Proposer (π, the trained agent) who holds a resource pool and must propose how to divide it, and a Responder (π_{env}, the static environment agent) who can accept, reject, or counter-propose. Both agents receive system prompts that establish their role, the pool size, the number of negotiation rounds, and the terminal markers. The full prompts are provided below.

Proposer System Prompt (π)

```
You are Player 1 (the Proposer) in a resource-splitting game.

There is a pool of $[pool_size] to divide between you and Player 2.
You start with the entire pool; Player 2 starts with nothing.  You
will negotiate over [num_rounds] total rounds (alternating turns).

IMPORTANT: If no agreement is reached by the end of [num_rounds]
rounds, BOTH players get NOTHING. Rejection destroys the entire pool.

On your final turn, you must either Accept or Reject (no
counter-proposals).

RULES:
- Propose splits, counter-propose, or accept the other player's
proposal
- When you reach an agreement, respond with exactly:  \boxed{AGREED:
split=$X} where X is YOUR share
- To reject and end the game (both get zero), respond with:  \boxed{NO
DEAL}
```

Responder System Prompt (π_{env})

```
You are Player 2 (the Responder) in a resource-splitting game.

There is a pool of $[pool_size] to divide between you and Player 1.
Player 1 holds the entire pool and will propose a split.  You will
negotiate over [num_rounds] total rounds (alternating turns).

IMPORTANT: If no agreement is reached by the end of [num_rounds]
rounds, BOTH players get NOTHING. Rejection destroys the entire pool.

On your final turn, you must either Accept or Reject (no
counter-proposals).

RULES:
- You can accept a proposal, reject it (ending the game), or
counter-propose
- When you reach an agreement, respond with exactly:  \boxed{AGREED:
split=$X} where X is the amount Player 1 receives
- To reject and end the game (both get zero), respond with:  \boxed{NO
DEAL}
```

F.2 OPENING MESSAGE

The Proposer starts each negotiation with a scripted opening message:

Proposer Opening Message

```
Hello!  We need to split $[pool_size] between us.  I hold the pool and
will make the first proposal.  If we can't agree, we both get nothing.
What do you think would be a fair split?
```

F.3 Data Source

Scenarios are procedurally generated from the cross-product of two parameters:

- **Pool size** (p): drawn from $\{100, 500, 1000, 5000, 10000\}$.
- **Number of rounds**: drawn from $\{4, 6, 8\}$.

This yields 15 unique configurations, which are cycled to fill the desired number of scenarios per training round. The environment design follows the Ultimatum Game formulation from NegotiationArena (Bianchi et al., 2024).

F.4 Reward Structure

The reward measures the Proposer's agreed share relative to a baseline of an equal split:

$$R(s_\tau, a_\tau, x_{\tau+1}) = \begin{cases} \dfrac{X}{p/2} & \text{if agreement with Proposer share } X \\ 0 & \text{if no agreement (\texttt{\textbackslash boxed\{NO DEAL\}})} \end{cases}$$

where p is the pool size. An equal split ($X = p/2$) yields a reward of 1.0. Taking the entire pool ($X = p$) yields a reward of 2.0. A no-deal outcome, where both players receive nothing, yields a reward of 0. Agreements are detected by the terminal marker \boxed{AGREED: split=$X}, and failed negotiations by \boxed{NO DEAL}.

G Deal or No Deal Details

G.1 Agent Prompts

Both agents receive system prompts that establish the available items, their private valuations, and the rules for signaling outcomes. The trained agent π sees only its own values; the environment agent π_{env} sees its own (different) values. Neither agent knows the other's valuations. The full prompts are provided below.

Agent 1 (π) System Prompt

```
You are negotiating to divide items with another person.  You each
have different private values for the items | the other person's
values are unknown to you.

The items are:  book=(count:X value:Y) hat=(count:X value:Y)
ball=(count:X value:Y)

Your goal is to maximize your total points.  You get points only for
items you receive.  All items must be allocated between the two of
you.

RULES:
- When you reach an agreement, respond with exactly:  \boxed{AGREED:
book=X hat=Y ball=Z} where X, Y, Z are the number of items YOU receive
- If no agreement can be reached, respond with:  \boxed{NO DEAL}
```

Agent 2 (π_{env}) System Prompt

```
You are negotiating to divide items with another person.  You each
have different private values for the items | the other person's
values are unknown to you.

The items are:  book=(count:X value:Y) hat=(count:X value:Y)
ball=(count:X value:Y)

Your goal is to maximize your total points.  You get points only for
items you receive.  All items must be allocated between the two of
you.

RULES:
- When you reach an agreement, respond with exactly:  \boxed{AGREED:
book=X hat=Y ball=Z} where X, Y, Z are the number of items YOU receive
- If no agreement can be reached, respond with:  \boxed{NO DEAL}
```

The two prompts are identical in structure but are instantiated with each agent's private values. For example, Agent 1 might see `book=(count:2 value:4)` while Agent 2 sees `book=(count:2 value:1)` for the same scenario.

G.2 Opening Message

Agent 1 (π) starts each negotiation with a scripted opening message that states the item counts without revealing any value information.

Opening Message

```
Hello!  We need to divide some items between us:  X books, Y hats, Z
balls.  Let's discuss how to split them fairly.  What items are you
most interested in?
```

G.3 Data Source

Scenarios are drawn from the Deal or No Deal dataset of Lewis et al. (2017), loaded from HuggingFace (`mikelewis0/deal_or_no_dialog`, `dialogues` configuration). The dataset contains approximately 5,000 negotiation dialogues collected via Amazon Mechanical Turk. Each scenario specifies three item types (books, hats, balls) with varying counts and two sets of private values, one per agent. The values are constructed so that each agent's total value across all items sums to 10, ensuring that both agents have equal maximum attainable points but different preferences over item types.

G.4 Reward Structure

When Agent 1 (π) signals `\boxed{AGREED: book=X hat=Y ball=Z}`, the reward is computed from the agent's private values:

$$R(s_\tau, a_\tau, x_{\tau+1}) = \begin{cases} \dfrac{\sum_{i \in \{\text{book,hat,ball}\}} \text{allocation}_i \times \text{value}_i}{5.0} & \text{if agreement} \\ 0 & \text{if no deal} \end{cases}$$

The numerator is the agent's total points from its allocated items. The denominator of 5.0 corresponds to the equal-split score (half of the 10 total points available to each agent), so a reward of 1.0 represents obtaining exactly half the maximum possible points. For example, if Agent 1 receives 2 books valued at 3 each, the raw score is 6 and the reward is $6/5.0 = 1.2$. The environment validates that allocated amounts do not exceed the available item counts; invalid allocations are treated as no deal ($R = 0$).

H CaSiNo Details

H.1 Agent Prompts

Both campers receive system prompts that establish the negotiation setting and their private priority orderings over three resource types (Food, Water, Firewood). The trained agent (π, Camper 1) and the static environment agent (π_{env}, Camper 2) receive prompts that are identical in structure but differ in the private priorities and participant-written reasons assigned to each camper. The full prompts are provided below.

Camper 1 System Prompt (π)

```
Imagine that you are on a camping trip.  Apart from some basic
supplies, you can collect additional food packages, water bottles
and firewood to make your trip better.  Since these are limited, you
will have to split them with your campsite neighbor.

There are 3 packages each of Food, Water, and Firewood (9 total) to
divide.

Your priorities:
- [Issue] is [High/Medium/Low] priority ([reason])
- [Issue] is [High/Medium/Low] priority ([reason])
- [Issue] is [High/Medium/Low] priority ([reason])

Points per package:  High priority=5, Medium=4, Low=3.  Maximize your
points.  Try hard to get as many items as you can!

RULES:
- When you reach an agreement, respond with exactly:  \boxed{AGREED:
Food=X Water=Y Firewood=Z} where X, Y, Z are the number of packages
YOU receive
- If no agreement can be reached, respond with:  \boxed{NO DEAL}
```

Camper 2 System Prompt (π_{env})

```
Imagine that you are on a camping trip.  Apart from some basic
supplies, you can collect additional food packages, water bottles
and firewood to make your trip better.  Since these are limited, you
will have to split them with your campsite neighbor.

There are 3 packages each of Food, Water, and Firewood (9 total) to
divide.

Your priorities:
- [Issue] is [High/Medium/Low] priority ([reason])
- [Issue] is [High/Medium/Low] priority ([reason])
- [Issue] is [High/Medium/Low] priority ([reason])

Points per package:  High priority=5, Medium=4, Low=3.  Maximize your
points.  Try hard to get as many items as you can!

RULES:
- When you reach an agreement, respond with exactly:  \boxed{AGREED:
Food=X Water=Y Firewood=Z} where X, Y, Z are the number of packages
YOU receive
- If no agreement can be reached, respond with:  \boxed{NO DEAL}
```

The bracketed fields are filled from the scenario: each camper's three issues are assigned High, Medium, and Low priority (one each), and the participant-written reason for each priority is included verbatim from the dataset.

H.2 Opening Message

Camper 1 (π) starts each negotiation with a scripted opening message:

Camper 1 Opening Message

```
Hi there!  Looks like we need to split up some camping supplies.  I
have some preferences about what I'd like.  How about you?  What are
you looking for most?
```

H.3 Data Source

Scenarios are drawn from the CaSiNo dataset (Chawla et al., 2021), which contains 1,030 campsite negotiation dialogues collected via Amazon Mechanical Turk. Each dialogue record includes private priority orderings (High, Medium, Low) over the three resource types for both participants, along with participant-written natural language reasons for each priority (e.g., "Water is High priority because I get dehydrated easily").

The dataset is hosted on HuggingFace (`kchawla123/casino`) as a single split. We manually partition it 85/15 into training and test sets. Each scenario is parameterized by the two campers' priority orderings and their associated reasons.

H.4 Reward Structure

The trained agent (π, Camper 1) reports its allocation using `\boxed{AGREED: Food=X Water=Y Firewood=Z}`, where X, Y, Z are the number of packages Camper 1 receives for each resource. Each allocation value must be between 0 and 3 (inclusive). Points are awarded based on the agent's private priority ordering:

- High priority issue: 5 points per package
- Medium priority issue: 4 points per package
- Low priority issue: 3 points per package

The reward is:

$$R(s_\tau, a_\tau, x_{\tau+1}) = \begin{cases} \dfrac{\sum_{i \in \{\text{Food,Water,Firewood}\}} n_i \cdot w_i}{18} & \text{if agreement} \\ 0 & \text{if no agreement} \end{cases}$$

where $n_i \in \{0, 1, 2, 3\}$ is the number of packages the agent receives for issue i, and $w_i \in \{5, 4, 3\}$ is the point weight corresponding to the agent's priority for that issue.

The normalization constant 18 equals the score from an equal split of all resources. Specifically, receiving 1.5 packages of each type yields $1.5 \times (5+4+3) = 18$ points, which is half the maximum possible score of 36 (obtained by claiming all 9 packages). This normalization ensures that an equal split maps to a reward of 1.0, while claiming all high-priority items yields rewards above 1.0.

If no agreement is reached, or if the agent outputs `\boxed{NO DEAL}`, the reward is 0.

I Job Interview Details

I.1 Agent Prompts

The Job Interview environment involves two agents: a Recruiter (π, the trained agent) who wants to minimize salary and holiday costs while optimizing discrete issue selections, and a Worker (π_{env}, the static environment agent) who wants to maximize salary and holidays. Both agents negotiate over five issues simultaneously. Each agent receives a system prompt with their role, the issue space, and a procedurally generated utility function specifying importance weights and option values. The full prompts are provided below.

Recruiter System Prompt (π)

```
You are a recruiter negotiating a job offer with a candidate.
You need to agree on 5 issues: Salary (20-51, in thousands),
Weekly Holiday (2-7 days), Workplace (Tokyo/Seoul/Beijing/Sydney),
Company (Google/Facebook/Apple/Amazon), and Position
(Engineer/Manager/Designer/Sales).

Your preferences (importance weights and option values):

- Salary (range 20-51, you prefer LOWER): importance = [weight]
- WeeklyHoliday (range 2-7, you prefer LOWER): importance = [weight]
- Workplace (importance = [weight]): Tokyo=[val], Seoul=[val],
Beijing=[val], Sydney=[val]
- Company (importance = [weight]): Google=[val], Facebook=[val],
Apple=[val], Amazon=[val]
- Position (importance = [weight], value depends on Company chosen):
  If Company=Google: Engineer=[val], Manager=[val], Designer=[val],
Sales=[val]
  If Company=Facebook: Engineer=[val], Manager=[val], Designer=[val],
Sales=[val]
  If Company=Apple: Engineer=[val], Manager=[val], Designer=[val],
Sales=[val]
  If Company=Amazon: Engineer=[val], Manager=[val], Designer=[val],
Sales=[val]

Your goal is to maximize your total score. The candidate has
different preferences. Try to find a deal that works for both of
you.

RULES:
- When you reach an agreement, respond with exactly: \boxed{AGREED:
Salary=X Holiday=Y Workplace=Z Company=W Position=P}
- If no agreement can be reached, respond with: \boxed{NO DEAL}
```

Worker System Prompt (π_{env})

```
You are a job candidate negotiating your offer with a recruiter.
You need to agree on 5 issues: Salary (20-51, in thousands),
Weekly Holiday (2-7 days), Workplace (Tokyo/Seoul/Beijing/Sydney),
Company (Google/Facebook/Apple/Amazon), and Position
(Engineer/Manager/Designer/Sales).

Your preferences (importance weights and option values):

- Salary (range 20-51, you prefer HIGHER): importance = [weight]
- WeeklyHoliday (range 2-7, you prefer HIGHER): importance = [weight]
- Workplace (importance = [weight]): Tokyo=[val], Seoul=[val],
Beijing=[val], Sydney=[val]
- Company (importance = [weight]): Google=[val], Facebook=[val],
Apple=[val], Amazon=[val]
- Position (importance = [weight], value depends on Company chosen):
  If Company=Google: Engineer=[val], Manager=[val], Designer=[val],
Sales=[val]
  If Company=Facebook: Engineer=[val], Manager=[val], Designer=[val],
Sales=[val]
  If Company=Apple: Engineer=[val], Manager=[val], Designer=[val],
Sales=[val]
  If Company=Amazon: Engineer=[val], Manager=[val], Designer=[val],
Sales=[val]

Your goal is to maximize your total score. The recruiter has
different preferences. Try to find a deal that works for both of
you.

RULES:
```

```
- When you reach an agreement, respond with exactly:  \boxed{AGREED:
Salary=X Holiday=Y Workplace=Z Company=W Position=P}
- If no agreement can be reached, respond with:  \boxed{NO DEAL}
```

The two prompts are structurally identical except for two differences: (1) the role description ("recruiter" vs. "job candidate"), and (2) the direction of preference for integer issues. The recruiter prefers LOWER values for Salary and WeeklyHoliday, while the worker prefers HIGHER values. Discrete issue preferences (Workplace, Company, Position) are drawn independently for each party, so they may or may not conflict.

I.2 Opening Message

The Recruiter starts each negotiation with a scripted opening:

Recruiter Opening Message

```
Hello!  I'm the recruiter and I'd like to discuss the terms of
your offer.  We need to agree on salary, weekly holiday, workplace
location, company, and position.  What matters most to you?
```

I.3 Data Source

Scenarios are adapted from the multi-issue negotiation framework of Yamaguchi et al. (2021). Each scenario is procedurally generated by sampling independent utility functions for both the recruiter and the worker, producing a unique Pareto frontier per scenario.

The five negotiation issues comprise two integer-valued issues and three discrete issues:

- **Salary** (integer, range 20-51): represents salary in thousands. The recruiter prefers lower values; the worker prefers higher.
- **WeeklyHoliday** (integer, range 2–7): days off per week. The recruiter prefers lower values; the worker prefers higher.
- **Workplace** (discrete, 4 options): Tokyo, Seoul, Beijing, Sydney.
- **Company** (discrete, 4 options): Google, Facebook, Apple, Amazon.
- **Position** (discrete, 4 options, dependent on Company): Engineer, Manager, Designer, Sales. The utility of each position option depends on which company is selected, creating a cross-issue dependency.

Utility functions are generated as follows. First, $N = 5$ random importance weights are drawn uniformly, normalized to sum to 1, scaled to the range $[0.1, 0.6]$, and re-normalized. For each discrete issue, random option values are drawn and min-max normalized to $[0, 1]$. For the Position issue, a separate set of option values is generated for each possible Company choice, producing a 4×4 table of position-company bias values (also min-max normalized to $[0, 1]$). This procedure generates an unlimited supply of unique scenarios.

I.4 Reward Structure

The reward is the raw recruiter utility score, computed as a weighted sum of normalized per-issue scores:

$$R(s_\tau, a_\tau, x_{\tau+1}) = \begin{cases} \sum_{i \in \mathcal{I}} w_i \cdot \text{score}_i & \text{if agreement} \\ 0 & \text{if no agreement} \end{cases}$$

where $\mathcal{I} = \{\text{Salary}, \text{WeeklyHoliday}, \text{Workplace}, \text{Company}, \text{Position}\}$ and w_i is the importance weight for issue i (with $\sum_i w_i = 1$). The per-issue scores are computed as follows:

- **Integer issues** (Salary, WeeklyHoliday): the recruiter prefers lower values, so $\text{score}_i = (\text{max} - \text{chosen})/(\text{max} - \text{min})$.

- **Discrete issues** (Workplace, Company): score_i equals the option value from the recruiter's utility function (already in $[0, 1]$).
- **Dependent issue** (Position): score_i equals the position-company bias value for the chosen (Company, Position) pair.

The reward lies in $[0, 1]$ and is directly interpretable as the fraction of maximum possible recruiter utility. A no-deal outcome (signaled by `\boxed{NO DEAL}`) yields a reward of 0. Invalid agreements (Salary outside $[20, 51]$, WeeklyHoliday outside $[2, 7]$, or unrecognized discrete choices) also receive a reward of 0.

Agreements are detected by the terminal marker `\boxed{AGREED: Salary=X Holiday=Y Workplace=Z Company=W Position=P}`, and failed negotiations by `\boxed{NO DEAL}`.

J Alliance Negotiation Details

J.1 Agent Prompts

Both agents receive system prompts that establish the negotiation issues, the available options per issue, their private score matrices, and their BATNA (Best Alternative to a Negotiated Agreement) thresholds. The trained agent π (Stakeholder 1) sees only its own scores and threshold; the environment agent π_{env} (Stakeholder 2) sees its own (different) scores and threshold. Neither agent knows the other's preferences. The full prompts are provided below.

Stakeholder 1 (π) System Prompt

```
You are a stakeholder negotiating a project agreement with another
party.  You each have private preferences over the options for each
issue.

The issues and options are:
Issue A (Infrastructure):  A1=Water-based facilities, A2=Land-based
facilities
Issue B (Environment):  B1=Minimal impact, B2=Moderate restoration,
B3=Full restoration
Issue C (Employment):  C1=Unlimited hiring, C2=Preference to locals,
C3=Partial union, C4=Full union
Issue D (Funding):  D1=Full federal loan, D2=Partial loan, D3=Minimal
loan, D4=No federal loan
Issue E (Compensation):  E1=Maximum compensation, E2=High
compensation, E3=Moderate compensation, E4=Low compensation, E5=No
compensation

Your private scores for each option:

Issue A (Infrastructure):  A1 (Water-based facilities):  X pts, A2
(Land-based facilities):  Y pts
Issue B (Environment):  B1 (Minimal impact):  X pts, B2 (Moderate
restoration):  Y pts, B3 (Full restoration):  Z pts
...

Your minimum acceptable score (BATNA threshold):  [threshold]
You should REJECT any deal that gives you fewer than this many points.

Your goal is to maximize your score while ensuring it stays above
your threshold.  The other party has different (likely opposing)
preferences.

RULES:
- When you reach an agreement, respond with exactly:  \boxed{AGREED:
A=1 B=2 C=3 D=1 E=4} using issue letters and option numbers
(1-indexed)
- If no agreement can be reached, respond with:  \boxed{NO DEAL}
```

Stakeholder 2's prompt is identical in structure but is instantiated with Stakeholder 2's private score matrix and BATNA threshold.

J.2 Opening Message

Stakeholder 1 (π) starts each negotiation with a scripted opening message:

Stakeholder 1 Opening Message

```
Hello!  We need to agree on the terms of this project.  There are
several issues to negotiate and I'm sure we can find something that
works for both of us.  Which issues are most important to you?
```

J.3 Data Source

The environment is adapted from the LLM-Deliberation framework of Abdelnabi et al. (2024). The five issue labels (Infrastructure, Environment, Employment, Funding, Compensation) and their option structures are drawn from the base game in that work, with $[2, 3, 4, 4, 5]$ options per issue respectively.

Scenarios are procedurally generated with anti-correlated score matrices. For each scenario, the total score budget of 100 points is allocated randomly across the five issues, and per-issue scores are distributed across options such that the two agents' score vectors have a correlation sampled uniformly from $[-1.0, -0.5]$. This ensures that the agents have opposing preferences without being perfectly adversarial.

BATNA thresholds are generated per difficulty level:

- **Easy**: threshold sampled from $[0.2, 0.35]$ of the feasible score range.
- **Medium**: threshold sampled from $[0.35, 0.55]$ of the feasible score range.
- **Hard**: threshold sampled from $[0.55, 0.70]$ of the feasible score range.

A feasibility check ensures that at least one deal satisfies both agents' thresholds; thresholds are relaxed if no feasible deal exists. The generator produces an unlimited number of unique scenarios.

J.4 Reward Structure

When Stakeholder 1 (π) signals `\boxed{AGREED: A=... B=... ...}`, the reward is computed from the agent's private score matrix:

$$R(s_\tau, a_\tau, x_{\tau+1}) = \begin{cases} \dfrac{\sum_{i \in \{A,B,C,D,E\}} \text{score}_i[\text{option}_i]}{50.0} & \text{if agreement and total score} \geq \text{threshold} \\ 0 & \text{if agreement but total score} < \text{threshold (below BATNA)} \\ 0 & \text{if no deal} \end{cases}$$

The numerator is the sum of Stakeholder 1's scores for the chosen option on each issue. The denominator of 50.0 corresponds to half of the maximum attainable score (100), so a reward of 1.0 represents obtaining half the total points. For example, if the agreed options yield a total score of 65 and the agent's threshold is 40, the reward is $65/50.0 = 1.3$. If the total score falls below the BATNA threshold, the reward is 0 regardless of the score, penalizing deals that are worse than the agent's outside option.

Agreements are detected by the terminal marker `\boxed{AGREED: A=1 B=2 C=3 D=1 E=4}`, and failed negotiations by `\boxed{NO DEAL}`.

K ARCHER ANALYSIS

K.1 OVERVIEW

ArCHer (Zhou et al., 2024) is an off-policy actor-critic method for multi-turn RL with language models. It learns response-level Q- and V-functions via temporal difference learning and updates the policy via a REINFORCE-style actor loss. We evaluate two variants under the same iterative deployment protocol as I-GRPO (4 rounds of 400 conversations each, with identical LoRA configurations and base models):

- **ArCHer** (original): uses a RoBERTa-based embedding model as the double critic, following the architecture of Zhou et al. (2024).
- **ArCHer-LC**: replaces the RoBERTa-based embedding critic with a generative model critic that shares the same language model backbone as the policy. The critic training hyperparameters are matched to I-GRPO's Q-function training (3 epochs, gradient accumulation of 4) rather than the original ArCHer critic schedule (50 epochs, no gradient accumulation).

Full hyperparameters for both variants are given in Appendix L.

K.2 LIMITATIONS OF THE ORIGINAL ARCHER IN MULTI-TURN SETTINGS

The original ArCHer algorithm was evaluated on GPT-2 (124M parameters) with short-response tasks such as 20 Questions (Zhou et al., 2024). When applied to our multi-turn negotiation environments, which produce substantially longer responses (50–200+ tokens), two properties of its architecture become problematic:

Lack of length normalization. ArCHer's actor loss uses the *sum* of per-token log-probabilities $\log \pi(a \mid s)$ across all tokens in a response. This sum scales linearly with response length, so longer responses produce proportionally larger gradient contributions. In our setting, this causes gradient instability: the policy can be pushed far from the base model in a single update step, leading to degradation across most (environment, model) configurations.

Critic context window. The original ArCHer uses a RoBERTa-base encoder (Liu et al., 2019) as its critic, which has a fixed context window of 512 tokens. In our multi-turn negotiation environments, conversation histories routinely exceed 1,000 tokens by mid-conversation, meaning the critic must truncate the input and is blind to the later turns where agreements are typically negotiated. This produces noisy Q- and V-estimates, which in turn yield unreliable advantages for the actor update.

K.3 ARCHER-LC: ADDRESSING THESE LIMITATIONS

ArCHer-LC addresses the critic context window limitation by replacing the RoBERTa-based embedding critic with a generative model critic that shares the same language model backbone as the policy. This allows the critic to process the full conversation history (up to the LLM's native context window of 8,192+ tokens) rather than truncating at 512 tokens, producing more reliable Q- and V-estimates for longer conversations. The critic backbone is kept frozen during training; only lightweight value heads are learned, keeping the additional compute cost modest.

In our experiments, ArCHer-LC exhibits substantially less degradation than the original ArCHer across environments and models. Full results are presented in Section 4.3.

K.4 EXPERIMENTAL NOTES

For a controlled comparison, we adapt both ArCHer variants to operate in the same batch-online regime as I-GRPO, with a smaller replay buffer (10,000 transitions vs. 100,000 in the original) and fewer iterations (4 vs. 2,000). The original ArCHer design includes 20 warmup iterations of critic-only training before actor updates begin; we use 0 warmup iterations. These adaptations are necessary for a matched-compute comparison. We use a single set of hyperparameters across all environments (Tables 6 and 7) and do not perform environment-specific tuning.

L HYPERPARAMETERS

All experiments use the same hyperparameters across all six environments.We report the full settings for each method below.

L.1 SHARED SETTINGS

All five methods share the following data-generation and model configuration:

Table 2: Shared hyperparameters across all methods.

Component	Hyperparameter	Value
Data generation	Scenarios per round	100
	Conversations per scenario (K)	4
	Data seed	42
	Max turns per conversation	10
Generation	Temperature	0.7
	Top-p	0.9
LoRA	Rank r	64
	Alpha α	64

L.2 I-GRPO

Iterative GRPO uses a multi-turn discount factor $\gamma = 0.9$ and alternates between Q-function fitting and GRPO policy optimization each round.

Table 3: I-GRPO hyperparameters.

Component	Hyperparameter	Value
Pipeline	Rounds	4
	Discount γ	0.9
Q-function	Learning rate	1×10^{-4}
	Epochs	3
	Batch size	8
GRPO	Learning rate	1×10^{-5}
	Epochs	1
	Per-device batch size	8
	Gradient accumulation steps	8
	Generations per prompt (G)	8
	KL penalty β	0.0

L.3 ST-GRPO

Single-turn GRPO is identical to I-GRPO except that the discount factor is set to $\gamma = 0.0$, so every turn in a conversation receives the same (undiscounted) episode return as its advantage signal. This ablates the multi-turn credit assignment provided by the Q-function.

L.4 MT-SFT

Multi-turn SFT uses the same iterative pipeline to generate rollouts each round, then finetunes on the highest-reward conversations via supervised learning. It uses $\gamma = 0.0$ for rollout selection (i.e., conversations are ranked by undiscounted episode return).

Table 4: ST-GRPO hyperparameters.

Component	Hyperparameter	Value
Pipeline	Rounds	4
	Discount γ	0.0
Q-function	Learning rate	1×10^{-4}
	Epochs	3
	Batch size	8
GRPO	Learning rate	1×10^{-5}
	Epochs	1
	Per-device batch size	8
	Gradient accumulation steps	8
	Generations per prompt (G)	8
	KL penalty β	0.0

Table 5: MT-SFT hyperparameters.

Component	Hyperparameter	Value
Pipeline	Rounds	4
	Discount γ	0.0
SFT	Learning rate	1×10^{-5}
	Epochs	1
	Per-device batch size	1
	Gradient accumulation steps	8

L.5 ARCHER (ORIGINAL)

ArCHer (Zhou et al., 2024) is an off-policy actor-critic method that learns a response-level Q-function via temporal difference (TD) learning. The critic is a RoBERTa-base embedding model that scores full conversation prefixes. The actor is updated via a REINFORCE-style loss using advantages computed from the critic's Q- and V-estimates. We adapt ArCHer to use the same iterative data-generation pipeline as the other methods: at each iteration, fresh rollouts are collected with the current policy and added to a replay buffer before critic and actor updates.

All experiments were run with 3 random seeds per configuration.

L.6 ARCHER-LC

ArCHer-LC is an adaptation of ArCHer that replaces the RoBERTa-based embedding critic with a generative model critic sharing the same language model backbone as the policy. The critic training hyperparameters are matched to I-GRPO's Q-function training schedule. All other settings (actor update, replay buffer, pipeline) remain identical to the original ArCHer variant.

All experiments were run with 3 random seeds per configuration.

Table 6: ArCHer hyperparameters.

Component	Hyperparameter	Value
Pipeline	Iterations	4
	Warmup iterations	0
	Discount γ	0.9
Rollouts	Rollouts per iteration	400
	Replay buffer capacity	10,000
Critic	Encoder	RoBERTa-base
	Learning rate	1×10^{-4}
	Epochs per iteration	50
	Batch size	8
	Target network EMA τ	0.1
Actor	Learning rate	1×10^{-5}
	Epochs per iteration	3
	Batch size	4
LoRA	Rank r	64
	Alpha α	64
Inference	Max model context length	32,768
	Max turns	10
	Temperature	0.7

Table 7: ArCHer-LC hyperparameters. Only the critic differs from ArCHer (Table 6); all other settings are identical.

Component	Hyperparameter	Value
Pipeline	Iterations	4
	Warmup iterations	0
	Discount γ	0.9
Rollouts	Rollouts per iteration	400
	Replay buffer capacity	10,000
Critic	Encoder	Policy LLM (frozen)
	Learning rate	1×10^{-4}
	Epochs per iteration	3
	Batch size	8
	Gradient accumulation steps	4
	Target network EMA τ	0.1
Actor	Learning rate	1×10^{-5}
	Epochs per iteration	3
	Batch size	4
LoRA	Rank r	64
	Alpha α	64
Inference	Max model context length	32,768
	Max turns	10
	Temperature	0.7