REWARD DESIGN WITH LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reward design in reinforcement learning (RL) is challenging since specifying human notions of desired behavior may be difficult via reward functions or require many expert demonstrations. Can we instead cheaply design rewards using a natural language interface? This paper explores how to make reward design easier by using a large language model (LLM) such as GPT-3 as a proxy reward function, where the user provides a textual prompt containing a *few examples* (few-shot) or a *description* (zero-shot) of desired behavior. Our approach leverages this proxy reward function in an RL framework. Specifically, users specify a prompt once at the beginning of training. During training, the LLM evaluates an RL agent's behavior against the desired behavior described by the prompt and outputs a corresponding reward signal. The RL agent then uses this reward to update its behavior. We evaluate whether our approach can train agents aligned with user objectives in the Ultimatum Game, matrix games, and the DEALORNODEAL negotiation task. In all three tasks, we show that RL agents trained with our framework are well-aligned with the user's objectives and outperforms RL agents trained with reward functions learned via supervised learning.

1 INTRODUCTION

Autonomous agents are becoming increasingly capable with the rise of compute and data. This underscores the importance for human users to be able to control what policies learning agents converge to and ensure the policies are aligned with their objectives. For instance, imagine training an agent to represent users in a salary negotiation. A working mother fighting for a livable wage may want their agent to be stubborn whereas a new hire looking to develop a good relationship with the company may want their agent to be more versatile.

Currently, users specify desired solutions by designing reward functions or providing large amounts of labeled data. Both approaches are challenging for different reasons. On one hand, designing reward functions is not an intuitive way for non-computer scientists to specify preferences. For instance, it isn't straightforward how to write a reward function for a "versatile" negotiator. Furthermore, designing reward functions that balance between ... — also known as the "reward design problem" — is notoriously difficult because agents are susceptible to reward hacking Amodei et al. (2016); Hadfield-Menell et al. (2017).

On the other hand, it is not enough to learn from a single example; for instance we cannot expect a negotiation agent to know how to negotiate in all settings from one example negotiation. We need large amounts of labeled data. Crucially, both of these approaches do not generalize well to new users who have different objectives — we would have to re-design our reward functions or re-collect data.

Our aim is to create an easier way for users to communicate their preferences — one where they do not need to be well-versed in the art of crafting functions and one where they do not need to give copious amounts of demonstrations.

> *Our key insight is that given examples (or a single phrase) of a user's objective, an LLM might be able to give an accurate instantiation of reward values on a test example because LLM s have been trained on a lot of context and are also great in-context learners.*

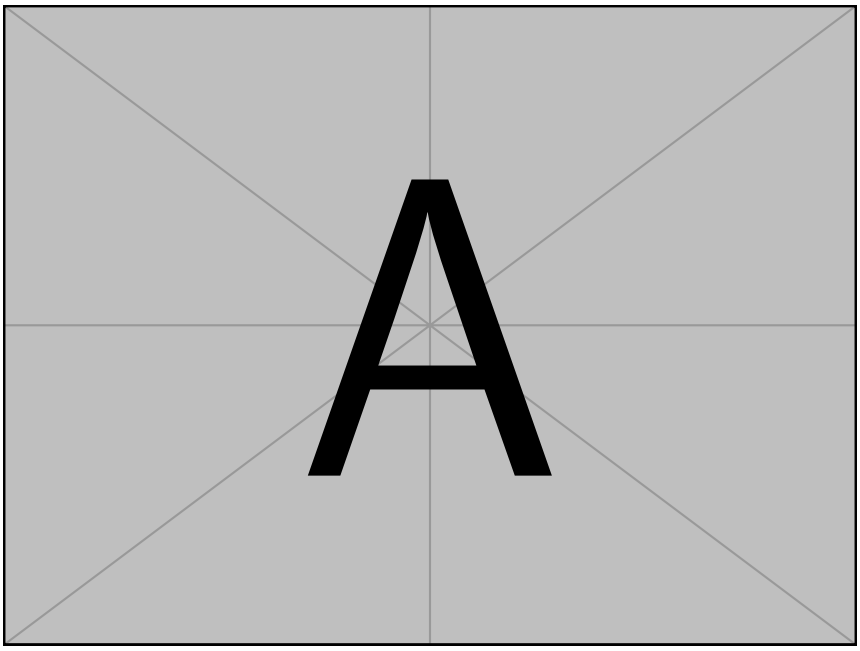

Figure 1: Caption

To this end, we explore how we can use LLM as a proxy for a reward function to train RL agents from user inputs. We use the word *proxy* because we are not using LLM to construct a reward function; rather we are using it to approximate the reward function at particular values. [TODO: maybe insert example here]. Our approach is simple: a user communicates their objective using natural language at the beginning of training. Objectives can be specified with a few examples when they are difficult to define (such as "versatility") or as a single phrase when they are well-known concepts (such as Pareto-optimality). During training, we parse an outcome of an RL episode into a string and ask LLM whether the outcome satisfies the user's objective. We then parse LLM's response into an integer and use this as the reward signal. The RL agent then updates its weights and continues training.

There are several advantages to using LLM as a proxy reward function: (1) users can specify their preferences in an intuitive manner using language; (2) we can leverage LLM's in-context learning abilities so that users only need to provide a handful of examples; (3) we do not need to re-design new reward functions for different objectives; and finally, (4) we can directly use the output of an LLM in an RL training loop. On the other hand, a potential disadvantage of using LLM is that it is unclear how much prompt design will be required to get reasonable behavior. The goal of this paper is to explore how feasible it is to use LLM as a reward function and understand when it makes sense to train with it.

Our contributions are as follows:

- We introduce the idea of using foundation models as a proxy reward function.
- We incorporate this proxy reward function into existing RL algorithms.
- We show that an LLM can successfully train objective-aligned RL agents using few-shot and zero-shot prompting in the Ultimatum Game, matrix games, as well as a negotiation task.
- We provide a discussion to evaluate when our approach will work best, where it fails, and areas for future work.

2 Related Work

3 Using LLM s as a Reward Signal

Our aim is to train an RL system using a language description of the goal. We formalize a task using a Markov Decision Process $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, \mathcal{R}_\theta, \rho_0, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition probability, and γ is the discount factor. $\mathcal{R}_\theta : S \times S \rightarrow \{0, 1\}$ is our modified reward function. It is conditioned on some LLMand takes in a goal string specified by the user as well as an outcome string that describes an RL episode. It then outputs a binary reward. While we focus on sparse reward signals in this work, we leave non-binary rewards for future work.

Our framework is depicted in Fig. 1 and outlined in Alg. **??**. Our algorithm takes as input a string that specifies the goal of the RL system. During trainintg, we roll out an episode and parse the outcome of the episode into a string. An outcome can describe the final state of the episode, the actions taken, or a trajectory that includes both states and actions over time. We use

4 Experiments

Two key advantages of our approach are that users can describe objectives that are hard to write as functions using language, and that the output of an LLM can be directly used within an RL training loop. However, in practice, it is unclear whether it is feasible to use LLMs during RL training. Furthermore, it's also unclear how much prompt-design will be required and whether LLMs will give sensible answers. In this section we investigate four questions: (Q0) Is it possible to use LLMs as a proxy reward in an RL training loop? (Q1) Can LLMs produce reward signals that are consistent with user objectives from a few examples (few-shot prompting)? (Q2) When preferences are well-known, can LLMs produce objective-consistent reward signals from a single phrase without any examples (zero-shot prompting)? (Q3) Can LLMs provide objective-consistent reward signals from examples (few-shot prompting) in more complicated domains?

We evaluate our approach on three domains: the Ultimatum Game, 2-player matrix games, and the DealOrNoDeal negotiation task Lewis et al. (2017). The Ultimatum Game and matrix games are a single timestep and serve as good toy domains. We illustrate LLM's abilities to quickly recognize user preferences (few-shot prompting) in the Ultimatum Game as well as LLM's ability to demonstrate its knowledge of well-known solution concepts in the matrix games (zero-shot prompting). The DealOrNoDeal task is a complex, multi-timestep domain where we evaluate LLM's ability to not only shape *what* solutions an RL agent converges to but *how* an RL agent converges to it (i.e., shape the style of trajectory). In all of our domains, we explain our prompt-tuning procedure and how we selected user preferences. We pick preferences that users may have according to prior literature in each domain. We did not run human studies to query users for preferences, but we leave this for future work. All our results are reported across 3 random seeds.

Evaluation Metrics. We evaluate our approach using the following metrics:

- **Labeling Accuracy**. We construct ground-truth reward functions for each domain. We report the mean accuracy of LLM's predictions *during RL training* with respect to the ground-truth reward functions.
- **RL Agent Accuracy**. *After RL training*, we evaluate RL agents' performance with respect to the ground truth reward functions. We report the mean accuracy of RL agents.

We also have domain-specific evaluation metrics that we explain in later sections.

Baselines.

- **SL**. A supervised learning model trained to predict reward signals using the same examples given to LLM. This baseline only applies to domains where we use few-shot prompting (Ultimatum Game, DealOrNoDeal). Please refer to [TODO] for details on training and model architecture.
- **Random**. Used for zero-shot prompting (matrix games). We replace the user's preference with a random string in the prompt.

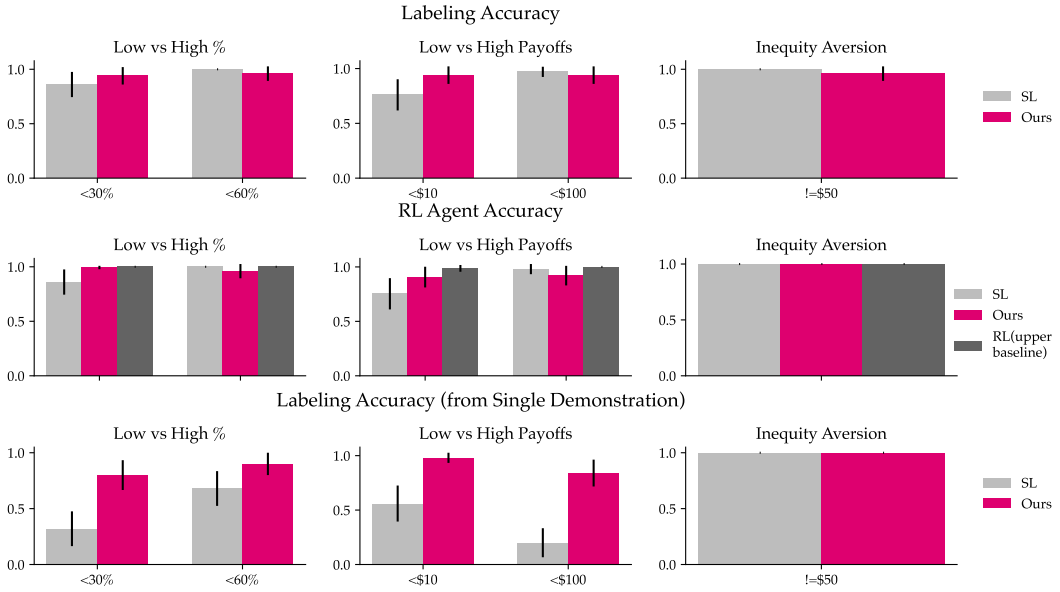

Figure 2: Ultimatum Game. (Top) Accuracy of reward signals provided by LLMand *during RL training. (Middle) Accuracy of RL agents after training. (Bottom) Accuracy of reward signals provided by LLMand during RL training from a single demonstration. LLMis able to maintain a high accuracy from a single demonstration.*

- **RL with ground truth reward functions**. RL agents trained with ground truth reward functions. We use this as an oracle.

4.1 (Q1) Agents converge to desired solutions with few-shot prompting

We first explore how feasible it is for LLMto learn user preferences from a few examples and train preference-consistent RL agents. We experiment in a single-timestep domain, the Ultimatum Game.

4.1.1 Task Description

The Ultimatum Game consists of two players, a Proposer and a Responder. A sum of money is endowed to the Proposer and they must propose a way to split the endowment with the Responder. The Responder can accept or reject the proposed split. If the Responder accepts, players receive money as per the split; if the Responder rejects, then both players get nothing. The game is a single-timestep.

Ground truth user preferences. A rational Responder would accept any proposal, even if it is unfair because getting something is better than getting nothing (in fact, this is a Nash Equilibrium of the game). However, prior work in behavioral economics shows that humans are willing to "punish" the Proposer by rejecting unfair proposals Vavra et al. (2018). For instance, a student may reject an unfair proposal only if she receives less than 30% of the endowment whereas a wealthy banker may reject if they receive less than 50%. We experiment with the following preferences:

- **Low vs High Percentages**. Users will reject proposals if they receive less than $\{30\%, 60\%\}$ of the endowment.
- **Low vs High Payoffs**. Users will reject unfair proposals if they receive less than $\{\$10, \$100\}$. They accept unfair proposals otherwise.
- **Inequity Aversion (Fehr & Schmidt (2010))**. Users will reject proposals if they do not receive exactly 50% of the endowment.

4.1.2 PROMPT TUNING

Template. A goal string consists of 10 examples of the Ultimatum Game. An example consists of the proposed split, the Responder's action, and the user's label of whether the Responder's action was desirable or undesirable. We do not explicitly mention the user's preferred threshold in the prompt. See Fig. **??** for an example.

Tuning Procedure. We randomly generated 10 training proposals used to prompt LLMand 50 test proposals used to evaluate LLM. For *Low vs High Percentages* and *Low vs High Payoffs*, we used the same set of training proposals across variants (i.e., same proposals for (30% and 60%) and ($10, $100)).

4.1.3 RESULTS

Labeling Accuracy. We evaluated our approach on our test set of 50 proposals. LLMachieves almost 100% labeling accuracy for every user preference, shown in Fig. 2(top). SL also performs well and is slightly more accurate than LLMfor three categories ($< 60\%$, $< \$100$, $!=\$50$). This result is not surprising, given that the decision boundary SL must learn for these binary decision tasks can easily be done using 10 training examples.

Instead, if we prompt LLMwith a single example followed by an explanation (see example in Fig. **??**), LLMmaintains a high accuracy whereas training SL with a single example hurts its accuracy. The accuracy for *Inequity Aversion* remains high because most of the labels are overwhelmingly negative.

RL Agent Accuracy. The accuracy of the trained RL agents mirror the labeling accuracy (middle of Fig. 2).

Summary. *LLMis able to in-context learn a user's preferences from a few examples and train an RL agent to converge to desired solutions. Importantly, we did not need to collect large amounts of new data or re-train our model for each new preference.*

4.2 (Q2) AGENTS CONVERGE TO DESIRED SOLUTIONS WITH ZERO-SHOT PROMPTING

When user preferences are well-known, can we prompt LLMwithout giving any demonstrations? Our hypothesis is that well-known preferences are more likely to be in-distribution, and thus LLMshould be able to train an RL agent to converge to those preferences without any demonstrations. Since we are zero-shot prompting, we do not use a SL baseline. Instead, we use a random baseline where we replace the well-known concept with a random string in our prompt.

4.2.1 TASK DESCRIPTION

We consider four 2-player normal-form matrix games (Battle of the Sexes, Stag Hunt, Chicken, and Prisoner's Dilemma). Each matrix game has four joint outcomes that have different solution concepts associated with them.

Ground truth user preferences. Although Pure Nash Equilibria are traditional solution concepts for normal form matrix games, users may prefer a solution for other properties. For instance, in Prisoner's Dilemma, users may prefer both agents to cooperate because they will maximize total welfare even though it is not a Nash Equilibrium. We experiment with four well-known solution concepts in matrix games:

- **Total Welfare**. Outcomes that achieve the greatest sum of player rewards.
- **Equality**. Outcomes that result in equal rewards between players.
- **Rawlsian Fairness**. Outcomes that maximize the minimum reward a player receives.
- **Pareto-Optimality**. Outcomes where the one of the corresponding rewards cannot be improved without lowering the other.

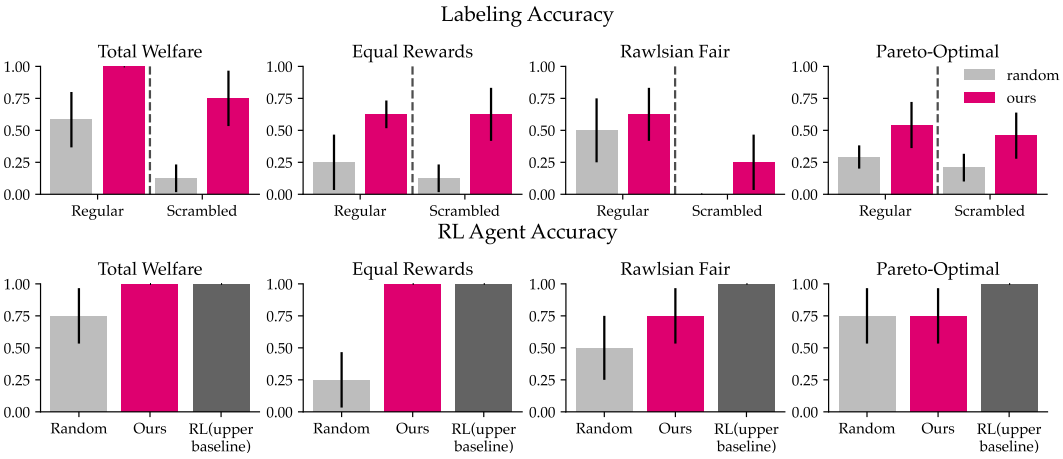

Figure 3: Matrix games. (Top) Accuracy of reward signals provided by LLMand a random baseline during RL training. We report results for both regular and scrambled versions of matrix games. (Bottom) Accuracy of RL agents after training.

4.2.2 Prompt-tuning Procedure

Template. Prompts for each solution concept are shown in Fig. **??**. Our prompt enumerates the outcomes of a matrix game and then asks LLMfor the outcome(s) that satisfy a solution concept. We do not mention the name of the matrix game in the prompt. We use chain-of-thought prompting by asking LLMto "think step-by-step" and provide a definition of the solution concept. To prevent any bias LLMmay have towards the order in which the outcomes of a matrix game are presented, we also randomly scramble the order of those outcomes (example shown in Fig. **??**).

Tuning Procedure. We tuned our prompt on the Battle of the Sexes matrix game to find one that gave us the best results. During evaluation, we kept our template the same for all of the matrix games.

4.2.3 Results

	Solution Concept	**Random Word**
Total Welfare	Total welfare is *the sum of the rewards of both players.*	sdvshk is *a function of the expected value of the game.*
Equality	Equality of rewards is *only possible if both players receive the same reward.*	sdvshk of rewards is *only possible if both players take the same action.*
Rawlsian Fairness	Rawlsian fairness is *defined as the maxmin value of the game, which is the minimum reward that the player could get assuming that the other player is maximizing their reward.*	sdvshk fairness is *defined as "the average reward over all possible play histories is the same for all players".*
Pareto-Optimality	An outcome is Pareto-optimal *if there is no other outcome that would make one player better off without making the other player worse off.*	An outcome is sdvshk *if it is Pareto optimal and individually rational.*

Table 1: Definitions of each solution concept given by LLM. Random replaces the solution concept in the prompt with a random string. LLMproduces mostly-correct definitions for each solution concept.

Labeling Accuracy. Given that each game can have many outcomes that satisfy a solution concept, we consider LLMto be 100% correct if it identifies all the correct outcomes. If LLMidentifies one out of two correct outcomes, then we give it a score of 0.5. If LLMidentifies a correct outcome and an incorrect one, we consider the answer to be incorrect and give it a score of 0. The top

row of Fig. 3 shows the average labeling accuracy for each solution concept across the four matrix games. LLMachieves higher labeling accuracy than the random baseline suggesting that LLMis able to apply its knowledge of well-known solution concepts. Scrambling the order of outcomes also lowers accuracy for most solution concepts. We hypothesize that this is because scrambling the order of outcomes makes the game more out-of-distribution. This result suggests that it not only is important for the solution concept to be in-distribution, but the task as well. Overall, LLMis able to accurately identify outcomes that satisfy a well-known solution concept better than a random baseline. However, LLMdoes not do this as robustly as few-shot prompting (see Sec. **??**). TODO: that task is OOD even though we don't mention the game naem

RL Agent Accuracy. We trained an RL agent to converge to the outcome LLMidentified as correct in the for the regular, non-scrambled games. If LLMsuggests more than one outcome satisfies a solution concept, we randomly pick one and train an RL agent to converge to that outcome. Therefore, LLMonly needs to identify one correct outcome for each solution concept instead of identifying all of them. Results are shown on the bottom of Fig. 3. LLMis able to reliably train agents for *Total Welfare* and *Equal Rewards*, but less reliably train *Rawlsian Fair* and *Pareto-Optimal*. The score for the random baseline is high for *Pareto-Optimal* because a large percentage of outcomes are Pareto-Optimal across games.

Does LLMReally *Understand* these Solution Concepts?. While it is difficult to determine if our results imply that LLMunderstands these solution concepts, we can investigate how well LLMreasons. Table 4.2.3 shows how LLMdefines each solution concept zero-shot. LLMis able to successfully recall the definitions for each solution concept except for *Rawlsian Fairness* – it gets it partially correct. However, LLMthen varied in its ability to reason whether an outcome of a game satisfies a solution concept. Table 4.2.3 also shows how LLMdefaults to defining each concept for a random word.

Summary. *LLMis able to zero-shot-identify well-known solution concepts and train an RL agent to converge to them. However, users may achieve more robust results with few-shot prompting.*

4.3 (Q3) Shaping the style of trajectories in more complex domains

We have shown that LLMcan train RL agents in simple, single-timestep domains. Can LLMtrain RL agents in complex, multi-timestep domains? We investigate this question in the DealOrNoDeal Negotiation task.

4.3.1 Task Description

In the DealOrNoDeal, an agent $Alice$ must come to an agreement with her partner Bob on the allocation of a set of objects (*books*, *hats*, and *balls*). Agents are shown a context, which include the counts of each item and each agent's private utilities over each item. In the original domain, agents get rewarded based on the agreed upon split and how much utility each item holds. If $Alice$ and Bob reach a disagreement, both agents get nothing. Please see [TODO] for more details on the game. We train $Alice$ using on-policy RL by negotiating against a fixed partner model, which we refer to as Bob. Please see [TODO] for more details on the domain and training.

Ground truth user preferences. We use LLMto train $Alice$ to negotiate in a style preferred by a user. Consider a scenario where $Alice$ represents a user in a negotiation. A single working mother fighting for higher pay may want $Alice$ to be stubborn whereas a diplomat negotiating with leaders from other countries may prefer a more versatile style. We experiment with the following negotiation styles inspired by previous literature Sycara et al. (1997); Caputo et al. (2019):

- **Versatile**. $Alice$ does not suggest the same proposal more than once.
- **Push-Over**. $Alice$ gets less points than Bob.
- **Competitive**. $Alice$ gets more points than Bob.
- **Stubborn**. $Alice$ repeatedly suggests the same proposal.

4.3.2 PROMPT-SELECTION PROCEDURE

Template. Goal strings for each style are shown in Fig. **??**. A goal string consists of three demonstrations. Each demonstration contains a negotiation, a question asking whether Alice negotiated in a particular style, and a yes or no answer followed by a short explanation.

Tuning Procedure. To create example negotiations, we randomly sampled negotiation contexts and had a vanilla RL agent negotiate with another agent using those contexts. We created a held-out test context used to train our RL agents.

4.3.3 RESULTS

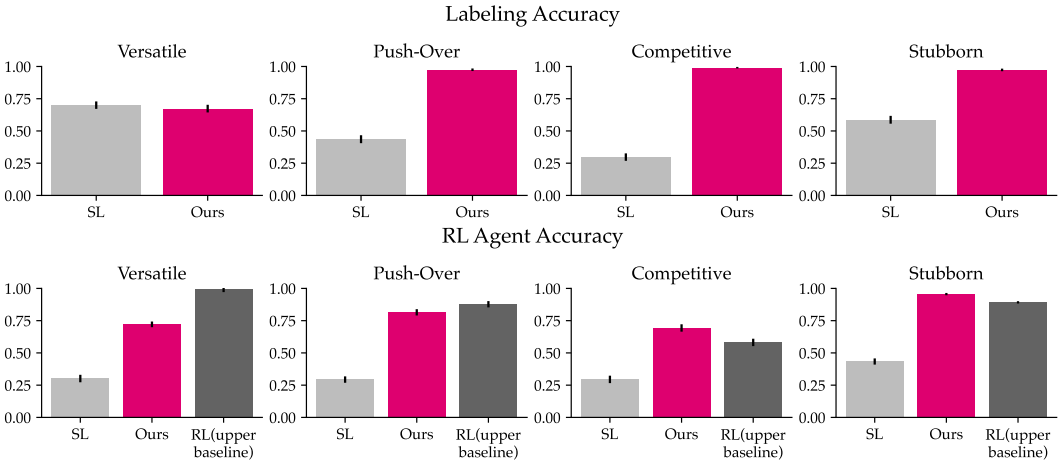

Figure 4: DEALORNODEAL. (Top) Accuracy of reward signals provided by LLMand SL during RL training. (Bottom) Accuracy of RL agents after training.

Labeling Accuracy. The top of Fig. 4 shows that LLMlabels similarly or more accurately than SL . For *Versatile*, both models perform similarly because SL learns to overwhelmingly predict a negative label (avg of 96% negative predictions) and the RL agent showed more negative examples of *Versatile* behavior (avg. of 70% ground truth negative labels). However, the large portion of negative examples prevents the agent from learning correct behavior as shown in the *Versatile* plot on the bottom of Fig. 4); here, we get a larger performance gap between LLMand SL . LLM's higher accuracy. This is not surprising given that we only provided three training examples for SL . LLM's high accuracy lends support to its ability to quickly in-context-learn a user's preferred style from a few examples with a one-sentence explanation.

RL Agent Accuracy. From the bottom of Fig. 4, we observe that results are consistent with each model's labeling accuracy. LLMachieves near perfect labeling accuracy for all styles except *Versatile*, and thus performs similarly to our upper baseline for those styles. LLMoutperforms our upper baseline for *Competitive* and *Stubborn*, but we attribute this to inconsistencies with RL training. With more seeds, we expect LLMslightly less accurate or equal to RL for *Competitive* and *Stubborn*.

Is there a qualitative difference in styles?. Example negotiations of agents trained with LLMfor each style are shown in Fig. **??**. To measure qualitative differences among styles, we looked at average advantage (Alice's rewards - Bob's rewards), diversity (percentage of Alice's utterances that are unique), and the agreement rate (percentage of negotiations that end in agreement). The results in Table 4.3.3 illustrate qualitative differences in styles. *Competitive* has the highest advantage whereas *Push-Over* has the lowest. *Versatile* achieves the highest diversity metric whereas *Stubborn* gets the lowest. Finally, *Push-Over* has a perfect agreement rate, suggesting that Alice has learned to be extremely agreeable.

Summary. *LLMcan be used to train RL agents in more complex, multi-timestep domains. Agents are not only able to align* what *state they converge to, but also* how *they accomplish a task.*

	Advantage	Diversity	Agreement Rate
Versatile	0.17 ± 0.91	0.99 ± 0.01	0.98 ± 1.89
Push-Over	-2.95 ± 0.64	0.82 ± 0.26	1.0 ± 0.0
Competitive	2.92 ± 0.64	0.74 ± 0.25	0.88 ± 6.5
Stubborn	1.36 ± 2.24	0.52 ± 0.1	0.82 ± 12.35

Table 2: Qualitative results describing negotiations produced by agents trained with LLM.

5 Further Analysis & Discussion

We have evidence that LLMcan be used to train preference-aligned RL agents. However, there are still many unanswered questions as to (1) what kinds of prompts work well and (2) whether our approach will work for all tasks and preferences. We discuss our insights to these questions below. Furthermore, we also experiment with zero-shot prompting on the rest of our domains (Ultimatum Game and DealOrNoDeal) to understand (3) how LLMzero-shot defines each preference and whether they are consistent with average human preferences reported in prior works.

5.0.1 (1) Discussion on Prompt Tuning

5.0.2 Does LLM

6 Limitations & Future Work

Finally, we discuss how well our approach works blah blah **Foundation Models**.

Sparse Rewards. The biggest limitation to our framework is that LLMonly specifies sparse, binary rewards. It is challenging to train RL agents with sparse rewards in more complex environments such as [] and [] . We leave the specification of non-sparse rewards up to future work.

Non-Spatial Tasks. We evaluated our approach on tasks that use natural language such as DealOrNoDeal, or tasks that can easily be converted to natural language. Spatial environments such as gridworlds, driving, or Mujoco environments are more challenging to describe using natural language. Using models that take in multi-modal input such as [] serve as a promising way to incorporate these environments.

Task Length. We experiment with tasks where an episode's horizon is a single timestep (Ultimatum Game and matrix games), or on average less than 10 timesteps long (DealOrNoDeal). Longer episodes may be more difficult to train since LLMmust retain more knowledge and we train our RL agent with sparse rewards. We believe developing methods to train LLMwith non-sparse rewards will also allow for longer-horizon or non-episodic tasks.

Multiagent Systems.

References

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Andrea Caputo, Oluremi B Ayoko, Nii Amoo, and Charlott Menke. The relationship between cultural values, cultural intelligence and negotiation styles. *Journal of Business Research*, 99:23–36, 2019.

Ernst Fehr and Klaus M Schmidt. On inequity aversion: A reply to binmore and shaked. *Journal of economic behavior & organization*, 73(1):101–108, 2010.

Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. Inverse reward design. *Advances in neural information processing systems*, 30, 2017.

Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, and Dhruv Batra. Deal or no deal? end-to-end learning for negotiation dialogues. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

Katia Sycara, Daniel Zeng, et al. Benefits of learning in negotiation. In *Proceedings of the AAAI National Conference on Artificial Intelligence. Menlo Park, California*, pp. 36–41, 1997.

Peter Vavra, Luke J Chang, and Alan G Sanfey. Expectations in the ultimatum game: distinct effects of mean and variance of expected offers. *Frontiers in psychology*, 9:992, 2018.

A Appendix

You may include other additional sections here.