Policy optimization to align the validity, coherence and efficiency of reasoning agents in multi-turn dialogues

Jeremy David Curuksu*
Amazon Web Services, Inc.
New York, NY, USA
curukj@amazon.com

Abstract

Reinforcement learning from human preferences can fine tune language models for helpfulness and safety, but does not directly address the fidelity and efficiency of reasoning agents in multi-turn dialogues. I propose a method to improve the validity, coherence and efficiency of reasoning agents by defining a reward model as a mapping between predefined queries and tools which can be applied to any custom orchestration environment. The reward model is used for policy optimization to fine tune the clarification fallback behavior and help the agent learn when best to ask for clarifications in multi-turn dialogues. This is demonstrated in several orchestration environments where after fine tuning with either proximal policy optimization or verbal reinforcement, the new policy systematically identifies the correct intents and tools in < 2 steps in over 99% of all sampled dialogues.

1 Introduction

Combining large languages models (LM) with policy optimization based on human preferences has recently led to new generations of chatbots showing human-level capabilities in helpfulness and safety, with each new release often massively better than previous LM generations. As shown in Bai et al. [2022], Touvron et al. [2023], Lee et al. [2023], by using different reward models for helpfulness and safety, reinforcement learning from human feedback (RLHF) and AI feedback (RLAIF) can be scaled to help find a better solution to the tradeoff between helpfulness and harmlessness. Direct policy optimization (DPO, Rafailov et al. [2024]) leads to similar improvements by tuning the policy directly from a preference dataset instead of encoding the dataset into a reward model for reinforcement learning. In all cases, the improved Pareto frontier between these two capabilities has been shown to result in more honest LM by reducing hallucination Bai et al. [2022], Touvron et al. [2023].

RLHF/RLAIF and DPO can fine tune general LM capabilities such as helpfulness and safety, but cannot directly fine tune the fidelity and efficiency of LM-based agents for generative retrieval, which is essential in the context of customer assistants orchestrating predefined sets of intents and possible actions to take. In this context, the agent must often navigate multi-turn dialogues that end when the original intent of the user is fulfilled. With new LM generations reaching human-level capabilities, it has become essential for customer assistants (e.g., Alexa, Siri) not just to increase their semantic capabilities, but to also behave logically and efficiently when the original intent of the user is ambiguous, in particular in cases where the intent is ambiguous *at first*, but simple enough that it can be disambiguated and solved with a few simple actions. For example, finding a good doctor in your area may require some clarification at first, but can be done nearly-perfectly after a few clarifications and a few clicks on the web, just as a human can do.

*Center for Data Science, New York University, NY 10011, USA *jeremy.cur@nyu.edu*

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

In this paper, I propose a method to directly optimize the fidelity and efficiency of a LM policy specifically for a predefined orchestration environment. The proposed method uses a general reinforcement learning formulation for sequential learning to fine tune LMs in multi-turn dialogues. This is in contrast to all RLHF, RLAIF, and DPO methods used in seminal papers from Stiennon et al. [2020], Ouyang et al. [2022], Bai et al. [2022], Touvron et al. [2023], Lee et al. [2023], Rafailov et al. [2024], which treat each turn in a dialogue as independent of each other and thus do not directly fine tune the strategic behavior of an agent in a sequential multi-turn dialogue. To directly fine tune the strategic reasoning capabilities of LMs in a dialogue, I define a reward model as a global (i.e., non-token based) mapping between some predefined queries and actions. This reward model is then used for policy optimization to fine tune the clarification fallback behavior in simulated multi-turn dialogues so the agent can learn when best to ask for clarifications. A separate LM generates prompt variation around each query, which increases robustness of the learned policies and enhances generalization. Despite the main limitation of this method which is the need to operate in a predefined environment, this method is very flexible as it can be applied to any customer assistant orchestrating a finite set of intents and possible actions i.e., most cases of practical interest.

To demonstrate the efficiency and flexibility of this method, I apply it on three different orchestration environments. The first is typical of customer assistants such as Alexa and Siri, where a custom set of user-intents, each associated to different prompts and slots (a.k.a. context data), is used to simulate multi-turn dialogues with a user. The agent is fine-tuned by proximal policy optimization (PPO, Schulman et al. [2017]) and learns selection policies across user intents and the clarification fallback. Some prompts and slots are voluntarily ambiguous and so the agent may learn to fall back on asking for clarification of intent.

In a second experiment, another LM agent interacts with the LM assistant and emulates the user. This second agent is also fine tuned by PPO and learns to select prompts that lead to higher reward, creating a collaborative environment. During PPO, a cooperation between the two agents emerges and ultimately reaches a Nash equilibrium Silver et al. [2018], as shown in the result section.

In a third experiment, I apply the method to a reasoning-acting agent (ReAct, Yao et al. [2022]), using different queries (create *vs.* analyze a picture, compute *vs.* execute an expression, retrieve one of two related but different publications) and corresponding tools, and a separate LM to generate prompt variation around each query. In this orchestration environment I apply "verbal" reinforcement learning (VRL) where the parameters of the LM are frozen and a simple look-up table of the reward accumulated for each prompt is appended to the agent's system prompt, and refreshed at every step during training. This verbal reinforcement learning was recently proposed in Shinn et al. [2024] and enables an LM to "reflect" on and learn from its past behavior, without ever changing its parameters. VRL was chosen in this third experiment to showcase the flexibility of the proposed method and its ability to fine tune reasoning agents using a wide range of learning algorithms.

In all orchestration environments tested, with either PPO or VRL algorithm, after fine tuning the policy systematically identifies the correct intents and tools in < 2 steps in over 99% of sampled dialogues. In contrast before fine tuning, in those orchestration environments, the starting policy took > 4 steps on average to disambiguate intents. This implies the new policy has learned to strategically fall back on asking for clarification of intent in ambiguous cases, instead of proceeding with a potentially wrong intent.

2 Policy optimization of reasoning agents in multi-turn dialogues

The general reinforcement learning formulation for sequential learning in multi-turn dialogues is presented in Fig. 1. An orchestration environment inherently defines a reward model by mapping a set of queries to a set of tools for a specific use case. The lack of determinism and need for fine tuning come from the nuances and potential ambiguities across the nearly infinite number of possible ways that a user, in a given natural language, may formulate each query. So it is assumed that a general pre-trained LM would not behave perfectly in such orchestration environment (as otherwise there would be no need for fine tuning). The imperfections are a direct consequence of potential ambiguity between prompts related to different intents and imperfect semantic parsing from the LM. In short, an orchestration setup is assumed to have inherent semantic pitfalls, which may lead to responses that are invalid, incoherent, or inefficient.

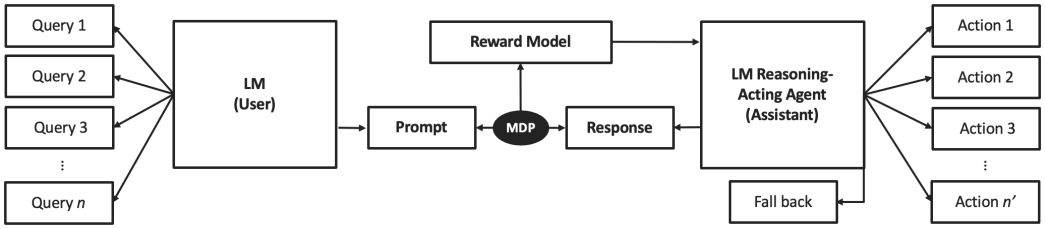


Figure 1: Fallback policy optimization of reasoning-acting agents. A custom mapping between n queries and n' actions defines the reward model used for fine tuning the agent.

The goal of the proposed method is to optimize the fallback of the agent so it systematically asks for clarifications when it is confused, instead of hallucinating or confidently making a wrong decision (e.g., invoking a wrong tool, proceeding with a wrong intent). Due to the deterministic nature of the query-to-action mapping in a predefined orchestration environment, an optimal fallback behavior can in principle be learned from an offline explore-exploit reinforcement learning algorithm.

The reward model shown in Fig 1 can be used as a preference model for policy optimization as in Christiano et al. [2017], Stiennon et al. [2020], Bai et al. [2022], Touvron et al. [2023], Lee et al. [2023], or any other reinforcement learning framework Sutton and Barto [2018]. Note that in Fig.1, n does not need to be equal to n' as long as each query is unambiguously mapped to a tool. In this paper I explore a sequential PPO framework learning across sequences of multi-turn dialogues Li et al. [2016], and verbal reinforcement learning Shinn et al. [2024]. This method directly fine tunes the performance of a LM in term of validity, coherence, and efficiency of dialogues, by learning where there are semantic pitfalls in this specific orchestration setup i.e., when to ask for clarifications in multi-turn dialogues. The details of the reward model, states, actions, and simulation setups, are described in the next section for three different orchestration environments.

3 Methodology

3.1 Proximal policy optimization of the clarification fallback

In a first experiment, a custom orchestration environment (summarized in Appendix A) was defined with a hierarchical modular organization. For example, the intents */pizza* and */dessert* are found within the intent */food*. These intents can be difficult to disambiguate because the prompts used by the user to invoke them may differ by only one or two tokens. Such prompts would benefit from asking for clarification of intent. Each intent is also associated with a variable number of required slots (a.k.a. context data) needed for fulfillment as in a typical Alexa-like customer assistant setup. The slots can be ambiguous too as some slots are 100% identical between different intents, such as *Are you looking for pick up or delivery* for all intents within */food*. This ambiguity was preserved to evaluate whether the agent could ultimately learn tradeoffs between efficiency and coherence i.e., fallback mitigation policies that start fulfilling slot data while asking for clarifications to determine the exact nature of the user's intent. As reported in the result section, the PPO agent did learn such emergent strategies by offline reinforcement learning without the need for human feedback.

An optimal dialogue can be defined as one which is (i) *valid*: the agent identifies the correct user intent, (ii) *coherent*: the agent solicits appropriate context data as required for fulfilling the user intent i.e., it does not solicit data *not* required for fulfillment, and (iii) *efficient*: the agent minimizes the time it takes to fulfill the user intent. In the first experiment, these goals are encoded explicitly into the following reward function:

$$r_t = \lambda_1 \, r_t^1 \, + \, \lambda_2 \, r_t^2 \, + \, \lambda_3 \, r_t^3$$

where λ_1, λ_2, λ_3 are preset weights (hyperparameters) associated to each reward component.

The component r_t^1 rewards the agent depending on whether it identifies the correct intent. The agent can select either of the user intents, or fall back on asking for clarification instead. If the agent guesses the user intent correctly, it receives a positive reward ($+5$). If the agent guesses the intent incorrectly, it receives a negative reward (-5). If the agent falls back on the clarification intent, it is neither rewarded nor penalized by this component i.e., $r_t^1 = 0$.

The component r_t^2 rewards the agent depending on whether it identifies the correct slots. The agent can prompt the user for either of the predefined slots (shown in Appendix A), or not prompt the user for any slot. If the agent solicits a valid slot, it receives a positive reward ($+5$). If the agent solicits slot-data not required for fulfillment, it receives a negative reward (-5). If the agent does not prompt the user for any slot, it is neither rewarded nor penalized by this component i.e., $r_t^2 = 0$.

The component r_t^3 systematically penalizes the agent by a negative value -1 at every step of the dialogue, so the agent is incentivized to close the dialogue quickly. By simultaneously mitigating all three goals of validity, coherence and efficiency, the agent tries to minimize the time it takes to identify the correct intent and solicit valid slots. Given some prompts are ambiguous, the agent may learn to fall back on asking for clarification at any time, for any prompt.

This explicit formulation of the three goals of validity, coherence and efficiency in the reward function is replaced by an implicit formulation in the third experiment (see section 3.3). The advantage of using an explicit formulation is to leverage the full potential of the reinforcement learning formalism for sequential learning with potentially delayed reward. To illustrate this, I decouple the parameters of the actor from the parameters of the LM. The LM is used exclusively for state representation (i.e., natural language embeding vector), and the actor is used exclusively to learn optimal sequential decision-making policies. Any LM encoder can be used to encode user prompts. I experimented with Amazon Titan and also a simpler Word2Vec model applied to the entire corpus of the environment defined by all queries, prompts, and slot requests (Appendix A). Policies learned were similar when using either encoder, probably due to the small size of the corpus.

In this orchestration environment the agent takes two actions simultaneously at every turn of a dialogue: it selects an intent or falls back on asking for clarification of intent, and asks the user to fill a slot or does not ask the user to fill a slot. As shown in appendix A, there are 6 possible intents and 6 possible slots across all intents.

To generate dialogues offline for PPO Schulman et al. [2017], intents are sampled uniformly at random and five possible prompts for each intent (pre-generated by Claude v3 in Amazon Bedrock) are also sampled uniformly at random. Each episode is a multi-turn dialogue between the user and the agent, given the user has chosen an intent and the agent needs to identify this intent (or ask for clarification) and fulfil it by soliciting valid slots.

3.2 PPO with Nash equilibrium

The above experiment was extended to a multi-agent setup, where two agents interact with each other: one agent emulates the assistant exactly as described in section 3.1, and one agent emulates the user. The key difference is when the assistant asks for clarification, or guesses the intent incorrectly, the user does not select a prompt randomly as in the experiment above, instead it learns to select prompts that lead to higher reward during offline PPO simulations.

Each agent becomes the *environment* from the perspective of the other agent. The behavior of each agent is learned directly by PPO from the sampled dialogues. Both agents aim to maximize the same reward function and thus a cooperation between the two agents is expected Silver et al. [2018], which can result in improved speed and coherence as confirmed in the result section. The multi-agent setup was created because in practice, a human user who regularly interacts with a chatbot tends to use words that the chatbot understands better. That is, a human user does not maintain a purely uniform and random choice of words as emulated in the first experiment. Results on the performance of learned policies in this multi-agent setup are reported for an identical orchestration environment as in experiment 1 (appendix A) and an identical number of sampled dialogues (30,000), to easily compare the two experiments, see result section.

3.3 Verbal reinforcement learning of the clarification fallback

Finally, the method was applied to an LM agent prompted for reasoning-acting (ReAct, Yao et al. [2022]). A ReAct agent engages in self-reflection before generating a response for the user: the LM first generates an internal thought, which is a decision to either use a tool or respond to the user. If it decides to use a tool, the tool is executed and returns some data which the agent then uses as additional context to generate another thought. The new thought is again a decision to either use a tool or respond to the user. This internal reasoning-acting loop continues until the agent decides it

has enough context data to respond to the user. A key requirement in ReAct Yao et al. [2022] is to predefine a finite set of tools. This is a perfect use case for our method. For this experiment, a set of 6 queries and 6 corresponding tools define the reward model (Appendix C). Queries were voluntarily defined to be ambiguous: create *vs.* analyze a picture, compute an expression with a calculator *vs.* execute an expression with an interpreter, retrieve a paper on RLHF *vs.* retrieve a paper on RLAIF.

A separate LM generates prompt variation for each query: 5 prompts were selected per query, again voluntarily selecting prompts that are very similar between queries. In particular, queries 3-4 and 5-6 share very similar prompts, see Appendix D for details.

In this ReAct orchestration environment, I apply "verbal" reinforcement learning where the parameters of the original LM are frozen, and a simple look-up table of the accumulated reward for each prompt variation is appended to the agent's system prompt, and refreshed at every step during training. VRL was recently proposed in Shinn et al. [2024] to align a LM by memorizing and reflecting on its past behavior instead of fine tuning its model parameters. VRL was chosen here to showcase the flexibility of the proposed method.

4 Results

4.1 Analysis of fallback proximal policy optimization in multi-turn dialogues

4.1.1 Detailed PPO simulation setup

The agent emulating the assistant was trained by simulating interactions $(s_t, a_t, r_{t+1}, s_{t+1})$ with users where t is the running number of turns in each dialogue. All simulated dialogues were generated using a custom OpenAI Gym environment and the PPO algorithm from the Intel Coach RL library on Amazon SageMaker, for a total number of interactions varying from 130,000 to 150,000 and representing a total of 30,000 dialogues. Each simulation was repeated 3 times and run on 36 CPUs using C5 9XL Amazon EC2 instances, taking approximately 5h each. An efficient exploration of state-action space was ensured by applying an ϵ-greedy exploration schedule: a search over different ϵ-schedules showed that linearly switching ϵ from 10% to 1% over the first 60,000 interactions led to the best results.

4.1.2 Analysis of User-Agent Interactions in Dialogues Simulated with PPO

Fig. 2 shows the evolution of accumulated reward in each dialogue. Three independent trials of 30,000 dialogues were produced to assess sensitivity to the initial random exploration. Fig. 2 suggests the PPO agent first explores the orchestration environment, then identifies a policy that mitigates the multiple learning goals of validity (identify the correct intent), coherence (solicit valid slots), and efficiency. In all trials, a transition is observed from a phase of random exploration in the first 10,000 episodes, where the accumulated reward ranges from -120 to $+20$, to a phase (final 5,000 dialogues) where the accumulated reward is between -40 and $+20$. This indicates that the PPO policy now systematically avoids certain behaviors when interacting with the user.

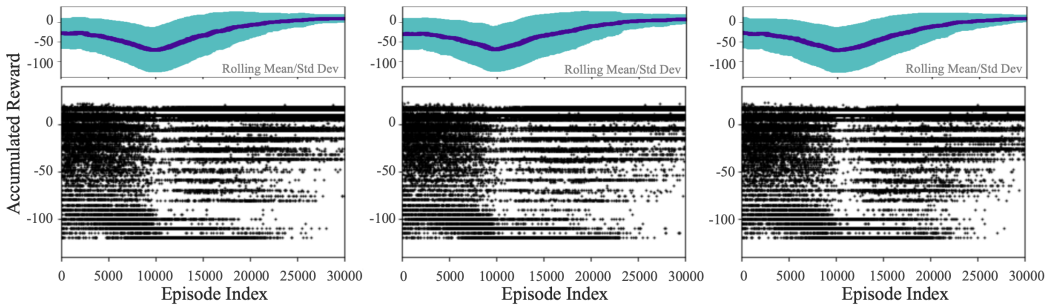


Figure 2: Accumulated rewards across 3×30,000 dialogues sampled with single-agent RL.

Table 1 shows the average proportion of *successful* dialogues (i.e., intent fulfilled) and number of steps in successful dialogues, per intent. Standard errors in parentheses were computed over the three

Table 1: Validity and efficiency of sampled dialogues.

	SINGLE AGENT		MULTI AGENT	
	0-5K	25-30K	0-5K	25-30K
% SUCCESS	68 (.8)	99 (.1)	67 (1.4)	100 (0)
NUMBER OF STEPS IN SUCCESSFUL DIALOGUES:				
INTENT 1	4.1 (.2)	1.0 (.0)	4.1 (.1)	1.0 (.0)
INTENT 2	4.2 (.1)	1.8 (.2)	4.1 (.1)	1.3 (.1)
INTENT 3	4.2 (.0)	1.6 (.2)	4.1 (.2)	1.3 (.0)
INTENT 4	4.1 (.1)	1.7 (.3)	4.3 (.0)	1.3 (.0)
INTENT 5	4.3 (.2)	1.7 (.3)	4.2 (.1)	1.3 (.0)
INTENT 6	4.4 (.1)	2.3 (.5)	4.3 (.1)	1.4 (.1)

trials shown in Fig. 2. In the first 5,000 dialogues, 68% are successful and take > 4 interactions to complete on average. In the final 5,000 dialogues, 99% are successful and these take at most 2 steps on average to infer the right intent. Fig. 2 confirms sub-optimal policies are still occasionally followed i.e., the PPO policy has not yet *fully* converged by the end of these simulations.

Appendix B shows a few examples sampled at the beginning and at the end of the simulations. In some dialogues, the agent learned to solicit a valid slot even when it felt back on asking the user for clarification of intent. The agent has thus learned some original policies which correctly infer slots required for fulfillment *even when the exact intent cannot yet be precisely determined*. This strategy spontaneously emerged by PPO and allows the agent to be more efficient i.e., to complete a dialogue with a smaller number of steps without sacrificing validity and coherence.

4.1.3 Nash Equilibrium in Dialogues Simulated with Multi-Agent PPO

When the assistant and the user are both PPO agents, a Nash equilibrium Silver et al. [2018] is expected because both agents try to maximize the *same* reward function. The only difference compared to the single-agent experiment above is that the user is also a PPO policy whose action is to select a prompt within the subset of five prompts predefined for a given intent.

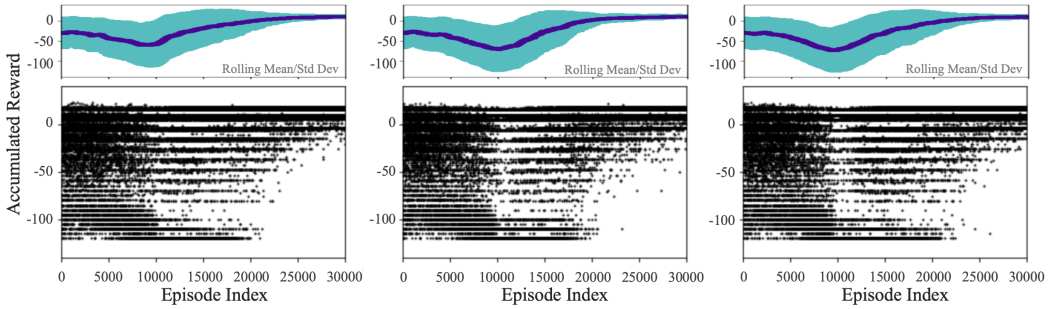


Figure 3: Accumulated rewards across 3×30,000 dialogues sampled with multi-agent RL.

As can be seen in Fig. 3, the agent converges toward optimal policies (reward in range -20 to $+20$) faster than in the single-agent simulations (Fig. 2) where the user selects prompts randomly. A second result observed in multi-agent simulations is that 100% of the policies followed by the end of the PPO simulations have a reward ranging between -20 and $+20$. The standard deviation observed in the final 5,000 dialogues is significantly smaller in Fig. 3 compared to Fig. 2. Table 1 also reports 100% of dialogues complete successfully. Thus, the cooperation of the user has eliminated the sub-optimal policies that were still occasionally observed in the end of the single-agent PPO simulations.

Table 1 indicates an improvement in *efficiency* for every intent. It takes 1.3 steps on average to infer the correct intent, compared to 1.8 steps in single-agent simulations. This makes sense because when the user sends prompts better understood by the agent, the agent less often needs to ask for clarifications. These results indicate that the PPO agent has identified reproducible policies to fall back on asking for clarification or infer the correct intent, solicit valid slots, and minimize the time it takes to fulfill intents.

4.2 Analysis of fallback policy optimization by verbal reinforcement learning

4.2.1 Detailed VRL simulation setup

In this third experiment, the ReAct agent and the agent used to generate prompt variations for each query both use Anthropic Claude v3 Sonnet available in Amazon Bedrock as backend LM. The ReAct agent was created using a custom ReAct system prompt and the LangChain react agent library. This agent was trained by simulating episodes of *reflections* which consist in a user prompt followed by reasoning-acting turns $(s_t, a_t, r_{t+1}, s_{t+1})$, where t is the total number of turns including the initial user prompt and every ensuing thought (s_t) and action (a_t) generated internally until the agent decides to respond to the user. Every time the agent selects a tool, which can happen more than once in an episode of reflection, a scalar numerical reward r_{t+1} $(+1$ for valid tool, -1 for invalid tool) is added to the current value of aggregated reward specifically for the original user prompt that initiated the agent's reflection. The latest value (aggregated reward) for every prompt is stored in a look up table and appended as context data to the agent system prompt. The agent is instructed to maximize reward. An ϵ-greedy exploration is reinforced by temporarily setting all values to 0 in 5% of user interactions. User queries and prompts are sampled uniformly at random for 10 epochs. The aggregated reward and number of clarification fallback are reported for each epoch for each prompt in the next section.

4.2.2 Analysis of Reasoning-Acting Generated in Reflections Simulated with VRL

Fig. 4 shows the evolution of accumulated reward and the number of times the ReAct agent decided to fall back on asking for a clarification instead of selecting a tool, in each loop of reflection initiated by a user prompt. Fig. 4 and Table 2 suggest the VRL agent first explores the orchestration environment, then identifies prompts which are too ambigous to identify the correct tool and are worth asking for clarification. For these ambiguous prompts, in the first few epochs the starting policy repeateadly select a wrong tool and does not ask for clarification (in other words, it hallucinates); in contrast, after only 10 epochs through all prompts and all queries, the accumulated reward has become sufficiently low (negative reward) relative to less ambiguous prompts, for the VRL agent to systematically fall back on asking for clarification instead of attempting to select a tool. In other words, the VRL agent learned to follow a policy which is honest, valid and efficient, because it never again attempts to select a tool when it receives a user prompt which it was never able to disambiguate in the past.

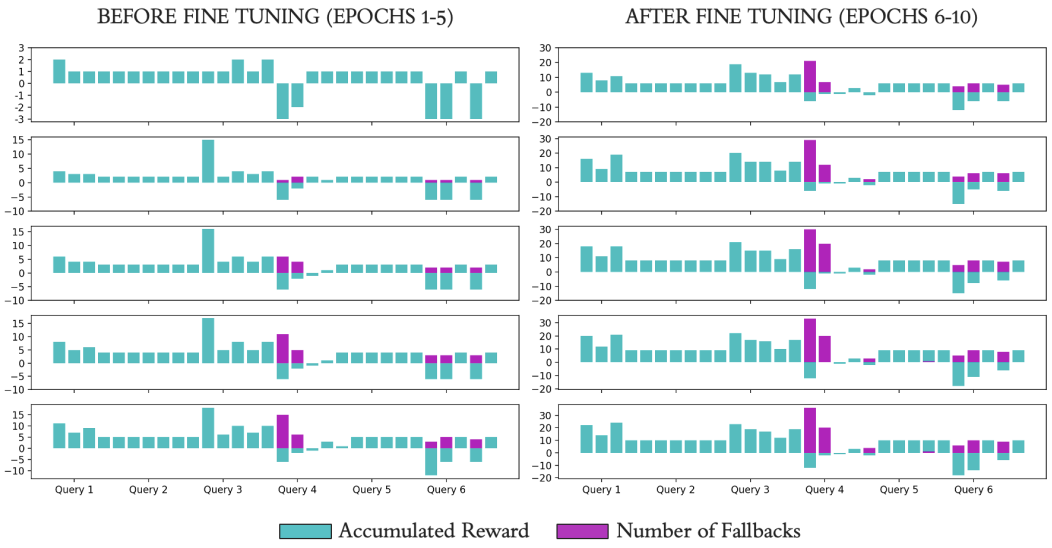


Figure 4: Accumulated reward and number of fallbacks across 10×30 episodes of reflections sampled with VRL. The x-axis corresponds to the five prompts generated by Claude for each of the six queries. Epochs 1-10 are listed from top to bottom.

Note that in the setup described in section 4.2.1, asking for a clarification leads to sample a different prompt for the same query, and as can be seen in Fig. 4 there are at most three ambiguous prompts out of five possible prompts for every query. On average, the correct tool is selected in < 2 steps across

Table 2: Validity and efficiency of sampled reflections.

	BEFORE FINE TUNING		AFTER FINE TUNING	
	REWARD	FALLBACK	REWARD	FALLBACK
QUERY 1	1.2 (.1)	0.0 (.0)	6.4 (1.7)	0.0 (.0)
QUERY 2	1.0 (.0)	0.0 (.0)	5.0 (.0)	0.0 (.0)
QUERY 3	1.4 (.2)	0.0 (.0)	9.4 (3.5)	0.0 (.0)
QUERY 4	-0.4 (1.7)	0.0 (.0)	-1.6 (2.1)	4.2 (5.0)
QUERY 5	1.0 (.0)	0.0 (.0)	5.0 (.0)	0.1 (.1)
QUERY 6	-1.4 (1.9)	0.0 (.0)	-2.0 (5.6)	2.2 (1.8)

all sampled episodes of reflections. These results are quantitatively similar to the results obtained in section 4.1 for different orchestration environments using PPO. Given the prompt variation is itself generated by Claude v3, this method can directly scale to a much larger number of prompt variations, queries and tools, in the context of superalignment. Future work will focus on analyzing the scalabilty of the VRL method, in particular when combined with predictive function approximation instead of the table lookup of memorized rewards used here to prove the concept.

5 Conclusion

Optimization of the clarification fallback policy was applied in three different reasoning-acting orchestration tasks, using PPO to fine tune model parameters, or using VRL to improve tool retrieval by reflecting on past sampled rewards. In all experiments, after fine tuning the new policy systematically identifies the correct intents and tools in < 2 steps in over 99% of sampled dialogues. In contrast before fine tuning, in those orchestration environments, the starting policy systematically hallucinated or took > 4 steps on average to disambiguate intents. This implies the new policy has learned to strategically fall back on asking for clarification of intent in ambiguous cases, instead of proceeding with a potentially wrong intent, and successfully completes dialogues after clarifying the user intent.

The method proposed in this paper defines a reward model as a mapping between predefined queries and actions. As a result, it can directly fine tunes the validity, coherence and efficiency of LM policies and reasoning agents in multi-turn dialogues, in contrast to RLHF, RLAIF and DPO methods in common usage. This method can be applied to any custom orchestration environment.

A limitation of the method is the need to operate in a predefined orchestration environment, where all possible queries and intents are well-defined, but human preferences have many facets including mixed and unclear intents, and intents changing over time. This may not represent a fundamental limitation to scale the proposed method because expressing mixed and unclear intents is typically a byproduct of imperfect communication, not an underlying goal or solution to a problem. In practice, most customer assistants (e.g., Alexa, Siri) do operate in the confine of a finite set of queries (problems to solve) and possible actions (solutions to solve problems). Disambiguating which action to take based on user prompts is precisely what the current method is designed to help with.

The VRL table lookup approach is unlikely to scale to very large numbers of queries and tools. Future work will focus on scaling the proposed method by replacing the table lookup approach by a function approximation approach, as typically done in the context of deep reinforcement learning.

A separate LM agent was used to generate prompt variation around each query, which increases robustness of the learned policies and enhances generalization. When the adversarial agent is a PPO policy incentivized to maximize the same reward as the assistant, the cooperation between the user and the assistant led to better dialogues. In practice, human users do not sample prompt uniformly at random so the cooperation may be more indicative of actual performance in online setups.

When the reasoning-acting agent needed to take two actions simultaneously, the agent learned an original policy that asks for valid context data even when the exact intent is not yet precisely known. This strategy spontaneoulsy emerged during the PPO simulations, without human feedback, and so showed potential for superalignment Bai et al. [2022], Lee et al. [2023]. The PPO policy found a way to increase efficiency without sacrificing quality and coherence.

Acknowledgments and Disclosure of Funding

The author thanks Amazon Web Services, Inc. (AWS) for compute resources.

References

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

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.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Kellie Lu, Thomas Mesnard, Colton Bishop, Victor Carbune, and Abhinav Rastogi. Rlaif: Scaling reinforcement learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*, 2023.

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

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, 2024.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, and T. Lillicrap. A general reinforcement learning algorithm that masters chess, shogi and go through self-play. *Science*, 362 (6419):1140–1144, 2018.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

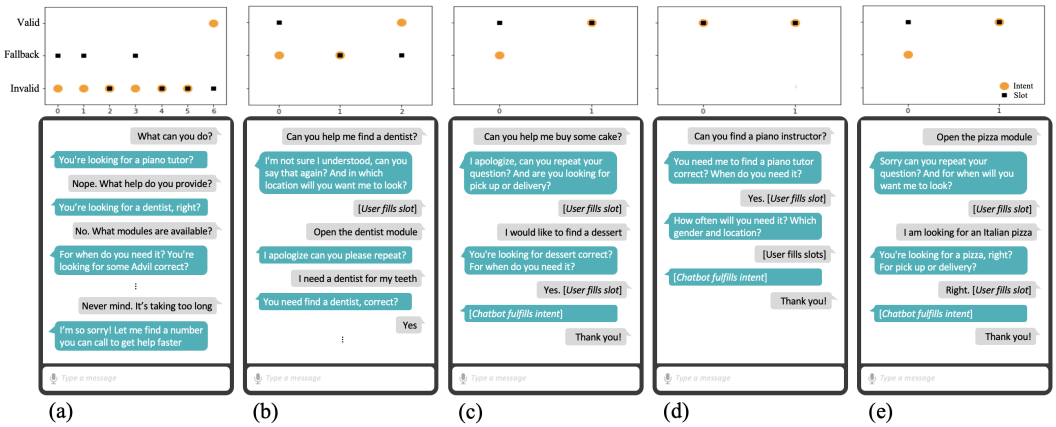
Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

A Appendix / supplemental material

INTENT	CORRESPONDING SLOTS
/NAVIGATION	NO SLOT
/TUTOR/PIANO	LOCATION, AGE, GENDER, WHEN, HOW OFTEN
/DOCTOR/DENTIST	LOCATION, WHEN
/PHARMACY/ADVIL	LOCATION, WHEN, HOW OFTEN, PICKUP OR DELIVERY
/FOOD/PIZZA	WHEN, PICKUP OR DELIVERY
/FOOD/DESSERT	WHEN, PICKUP OR DELIVERY
/CLARIFICATION	NO SLOT

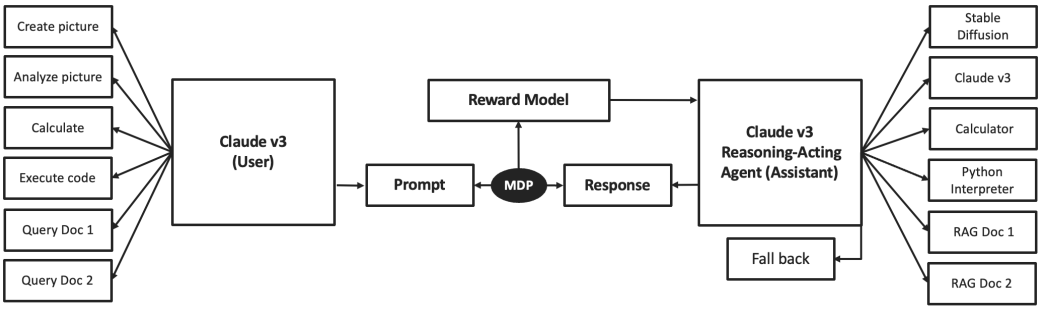
A Appendix A.

Map of intents and slots defining the custom orchestration environment in the two PPO fine tuning experiments (sections 3.1-3.2). The complete JSON file mapping all prompts to all intents and slot request (resulting in approximately 1230 possible combinations of prompts and slot requests for the RL agent to explore i.e., 7 intents $\times$ 5 prompts $\times$ 7 slot requests $\times$ 5 slots) is available upon request to the author.



B Appendix B.

Sample of user-PPO agent interactions before training (a), after an optimal policy has been identified by single agent PPO (b, c), and after an optimal policy has been identified by multi-agent PPO (d, e). The validity of every action taken at every step by the PPO agent is shown above each sampled dialogue.



C Appendix C.

Map of queries and tools in the custom orchestration environment used to define the reward model for VRL in the third experiment (section 3.3).

D Appendix D.

Prompts generated using Claude v3 Sonnet in Amazon Bedrock for all queries in the custom orchestration environment used for the VRL experiment (section 3.3). Most prompts were truncated to fit into the paper format; a JSON file mapping complete prompts (i.e., not truncated) for all queries is available upon request to the authors.

```
{"query": 1,
"prompts": [
"Please generate an image depicting several dogs running on a beach",
"I would like you to create a picture with multiple dogs playing on the...",
"Can you make an image showing a few dogs running and playing on the...",
"Create a drawing of multiple canines racing across the sand close to...",
"Produce an illustration displaying numerous dogs sprinting near the tide..."
]},
{"query": 2,
"prompts": [
"On the given picture, determine the number of dogs that are running on...",
"Please examine the provided image and count the quantity of dogs that...",
"Can you inspect the shown illustration and enumerate how many canines...",
"I need you to visually analyze the displayed graphic and tally the...",
"Kindly check the presented picture and calculate the total number of..."
]},
{"query": 3,
"prompts": [
"Calculate the result of the mathematical expression: 5 plus 5 plus 10",
"Can you determine the solution when 5 is added to 5 and then 10 is added...",
"I would like you to compute the total of 5 added to 5, with 10 added to...",
"Find the end quantity when 5 plus 5 is summed with 10",
"Derive the final value when 5 plus 5 plus 10 are combined together..."
]},
{"query": 4,
"prompts": [
"Calculate the result of the expression 5 plus 5 plus 10.",
"With i initially 0, iterate four times over this loop: i = i + 5, and tell...",
"Let i begin at 0. Repeat this process four times: add 5 to the current...",
"Initialize i to 0. Perform the following operation four times: i = i + 5. ...",
"With i initially assigned the value 0, cycle through this loop four..."
]},
{"query": 5,
"prompts": [
"What process did Anthropic use to adapt and adjust the harmful...",
"Please describe Anthropic's methodology to specialize and optimize the...",
"Can you explain the technique Anthropic leveraged to refine and enhance...",
"I would like you to delineate the approach taken by Anthropic to tune ...",
"How did Anthropic fine tuned the toxicity of their LLM?"
]},
{"query": 6,
"prompts": [
"What methods did Anthropic utilize to expand and amplify the precision...",
"Please discuss Anthropic's tactics to broaden and multiply the selective...",
"Can you clarify the procedures used by Anthropic to scale up and increase...",
"Describe how Anthropic was able to grow and augment the customized...",
"How did Anthropic scaled the fine tuning of the toxicity of their LLM?"
]}
```