Deep Reinforcement Learning from Human Preferences

Paul F Christiano
OpenAI
paul@openai.com

Jan Leike
DeepMind
leike@google.com

Tom B Brown
nottombrown@gmail.com

Miljan Martic
DeepMind
miljanm@google.com

Shane Legg
DeepMind
legg@google.com

Dario Amodei
OpenAI
damodei@openai.com

什么是强化学习

- 马尔科夫决策过程

$$(S_t, A_t, R_t, P_t)$$

S_t：环境状态

A_t：智能体的动作

R_t：奖励函数，取决于当前状态与动作

策略：状态到动作之间的映射

$P_t = p(S_{t+1} = s' | S_t = s, A_t = a)$：状态转移概率

固定且未知

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

马尔科夫特性

下一时刻的状态与奖励只取决于当前时刻的状态与动作——
——方便智能体对环境状态的变化规律进行推理

什么是强化学习

➢ 收益与价值函数

◆ **奖励：** $R_t = R(s_t, a_t)$ **短期利益**

◆ **收益：** $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ **长远利益**

◆ **动作-状态价值函数（Q值）：** $q_\pi(s,a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$

智能体的目标为学习到一个最优策略π^*，使智能体在任意状态下的价值最大

◆ **最优动作-状态价值函数（最优Q值）：** $q^*(s,a) \doteq \max_\pi q_\pi(s,a)$

$$a^*(s) = \max_a q^*(s,a)$$

学习目标 **求解最优策略π^*等价于求解最优Q值q^***

Problems

- Recent success in scaling reinforcement learning (RL) to large problems has been driven in domains that have a well-specified reward function. But many tasks involve goals that are complex, poorly-defined, or hard to specify.
- Suppose that we wanted to use RL to train a robot to clean a table or scramble an egg. We could try to design a simple reward function that approximately captures the intended behavior, but this will often result in behavior that optimizes our reward function without actually satisfying our preferences.
- This difficulty underlies recent concerns about misalignment between our values and the objectives of our RL systems

Motivation

- If we could successfully communicate our actual objectives to our agents, it would be a significant step towards addressing these concerns.
- An approach is to allow a human to *provide feedback on our system's current behavior and to use this feedback to define the task*. In principle this fits within the paradigm of reinforcement learning, but using human feedback directly as a reward function is prohibitively expensive for RL systems that require hundreds or thousands of hours of experience.
- In order to practically train deep RL systems with human feedback, we need to decrease the amount of feedback required by several orders of magnitude.

Motivation

- Our approach is to *learn a reward function from human feedback and then to optimize that reward function*. We desire a solution to sequential decision problems without a well-specified reward function that:
- 1. enables us to solve tasks for which we can only *recognize* the desired behavior, but not necessarily demonstrate it,
- 2. allows agents to be taught by non-expert users,
- 3. scales to large problems, and
- 4. is economical with user feedback.

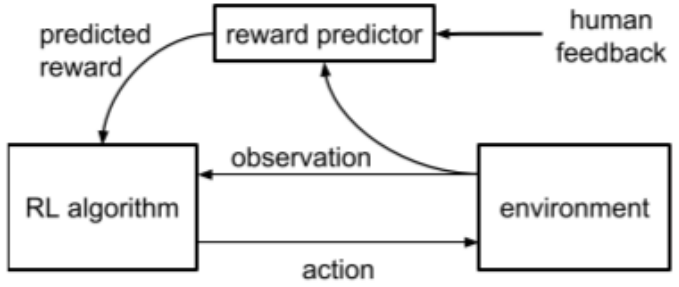


Figure 1: Schematic illustration of our approach: the reward predictor is trained asynchronously from comparisons of trajectory segments, and the agent maximizes predicted reward.

Preliminaries

We consider an agent interacting with an environment over a sequence of steps; at each time t the agent receives an observation $o_t \in \mathcal{O}$ from the environment and then sends an action $a_t \in \mathcal{A}$ to the environment.

In traditional reinforcement learning, the environment would also supply a reward $r_t \in \mathbb{R}$ and the agent's goal would be to maximize the discounted sum of rewards. Instead of assuming that the environment produces a reward signal, we assume that there is a human overseer who can express preferences between *trajectory segments*. A trajectory segment is a sequence of observations and actions, $\sigma = ((o_0, a_0), (o_1, a_1), \ldots, (o_{k-1}, a_{k-1})) \in (\mathcal{O} \times \mathcal{A})^k$. Write $\sigma^1 \succ \sigma^2$ to indicate that the human preferred trajectory segment σ^1 to trajectory segment σ^2. Informally, the goal of the agent is to produce trajectories which are preferred by the human, while making as few queries as possible to the human.

Evaluate algorithms' behavior

Quantitative: We say that preferences $\succ$ are *generated by* a reward function[1] $r : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ if

$$\left((o_0^1, a_0^1), \ldots, (o_{k-1}^1, a_{k-1}^1)\right) \succ \left((o_0^2, a_0^2), \ldots, (o_{k-1}^2, a_{k-1}^2)\right)$$

whenever

$$r(o_0^1, a_0^1) + \cdots + r(o_{k-1}^1, a_{k-1}^1) > r(o_0^2, a_0^2) + \cdots + r(o_{k-1}^2, a_{k-1}^2).$$

If the human's preferences are generated by a reward function r, then our agent ought to receive a high total reward according to r. So if we know the reward function r, we can evaluate the agent quantitatively. Ideally the agent will achieve reward nearly as high as if it had been using RL to optimize r.

Qualitative: Sometimes we have no reward function by which we can quantitatively evaluate behavior (this is the situation where our approach would be practically useful). In these cases, all we can do is qualitatively evaluate how well the agent satisfies to the human's preferences. In this paper, we will start from a goal expressed in natural language, ask a human to evaluate the agent's behavior based on how well it fulfills that goal, and then present videos of agents attempting to fulfill that goal.

Method

At each point in time our method maintains a policy $\pi : \mathcal{O} \to \mathcal{A}$ and a reward function estimate $\hat{r} : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$, each parametrized by deep neural networks.

These networks are updated by three processes:

1. The policy π interacts with the environment to produce a set of trajectories $\{\tau^1, \ldots, \tau^i\}$. The parameters of π are updated by a traditional reinforcement learning algorithm, in order to maximize the sum of the predicted rewards $r_t = \hat{r}(o_t, a_t)$.
2. We select pairs of segments $\left(\sigma^1, \sigma^2\right)$ from the trajectories $\{\tau^1, \ldots, \tau^i\}$ produced in step 1, and send them to a human for comparison.
3. The parameters of the mapping $\hat{r}$ are optimized via supervised learning to fit the comparisons collected from the human so far.

Optimizing the Policy

- We use advantage actor-critic (A2C) to play Atari games, and trust region policy optimization (TRPO) to perform simulated robotics tasks.

Preference Elicitation

The human overseer is given a visualization of two trajectory segments, in the form of short movie clips. In all of our experiments, these clips are between 1 and 2 seconds long.

The human then indicates which segment they prefer, that the two segments are equally good, or that they are unable to compare the two segments.

The human judgments are recorded in a database $\mathcal{D}$ of triples $\left(\sigma^1, \sigma^2, \mu\right)$, where σ^1 and σ^2 are the two segments and μ is a distribution over $\{1, 2\}$ indicating which segment the user preferred. If the human selects one segment as preferable, then μ puts all of its mass on that choice. If the human marks the segments as equally preferable, then μ is uniform. Finally, if the human marks the segments as incomparable, then the comparison is not included in the database.

Fitting the Reward Function

We can interpret a reward function estimate $\hat{r}$ as a preference-predictor if we view $\hat{r}$ as a latent factor explaining the human's judgments and assume that the human's probability of preferring a segment σ^i depends exponentially on the value of the latent reward summed over the length of the clip:[3]

$$\hat{P}\left[\sigma^1 \succ \sigma^2\right] = \frac{\exp \sum \hat{r}\left(o_t^1, a_t^1\right)}{\exp \sum \hat{r}(o_t^1, a_t^1) + \exp \sum \hat{r}(o_t^2, a_t^2)}. \tag{1}$$

We choose $\hat{r}$ to minimize the cross-entropy loss between these predictions and the actual human labels:

$$\text{loss}(\hat{r}) = -\sum_{(\sigma^1,\sigma^2,\mu)\in\mathcal{D}} \mu(1) \log \hat{P}\left[\sigma^1 \succ \sigma^2\right] + \mu(2) \log \hat{P}\left[\sigma^2 \succ \sigma^1\right].$$

- It can be understood as equating rewards with a preference ranking scale analogous to the famous Elo ranking system developed for chess. Just as the difference in Elo points of two chess players estimates the probability of one player defeating the other in a game of chess, the difference in predicted reward of two trajectory segments estimates the probability that one is chosen over the other by the human.

Selecting Queries

- We decide how to query preferences based on an approximation to the uncertainty in the reward function estimator.
- We sample a large number of pairs of trajectory segments of length k, use each reward predictor in our ensemble to predict which segment will be preferred from each pair, and then select those trajectories for which the predictions have the highest variance across ensemble members.

Experimental Setups

- We interface with MuJoCo and the Arcade Learning Environment.
- Reinforcement Learning Tasks with Unobserved Rewards:

- the agent learns about the goal of the task only by asking a human which of two trajectory segments is better, instead of the true reward.
- feedback is provided by contractors who are given a 1-2 sentence description of each task before being asked to compare several hundred to several thousand pairs of trajectory segments for that task

For comparison, we also run experiments using a synthetic oracle whose preferences over trajectories exactly reflect reward in the underlying task. That is, when the agent queries for a comparison, instead of sending the query to a human, we immediately reply by indicating a preference for whichever trajectory segment actually receives a higher reward in the underlying task[4]. We also compare to the baseline of RL training using the real reward. Our aim here is not to outperform but rather to do nearly as well as RL without access to reward information and instead relying on much scarcer feedback. Nevertheless, note that feedback from real humans does have the potential to outperform RL (and as shown below it actually does so on some tasks), because the human feedback might provide a better-shaped reward.

Experimental Results #1

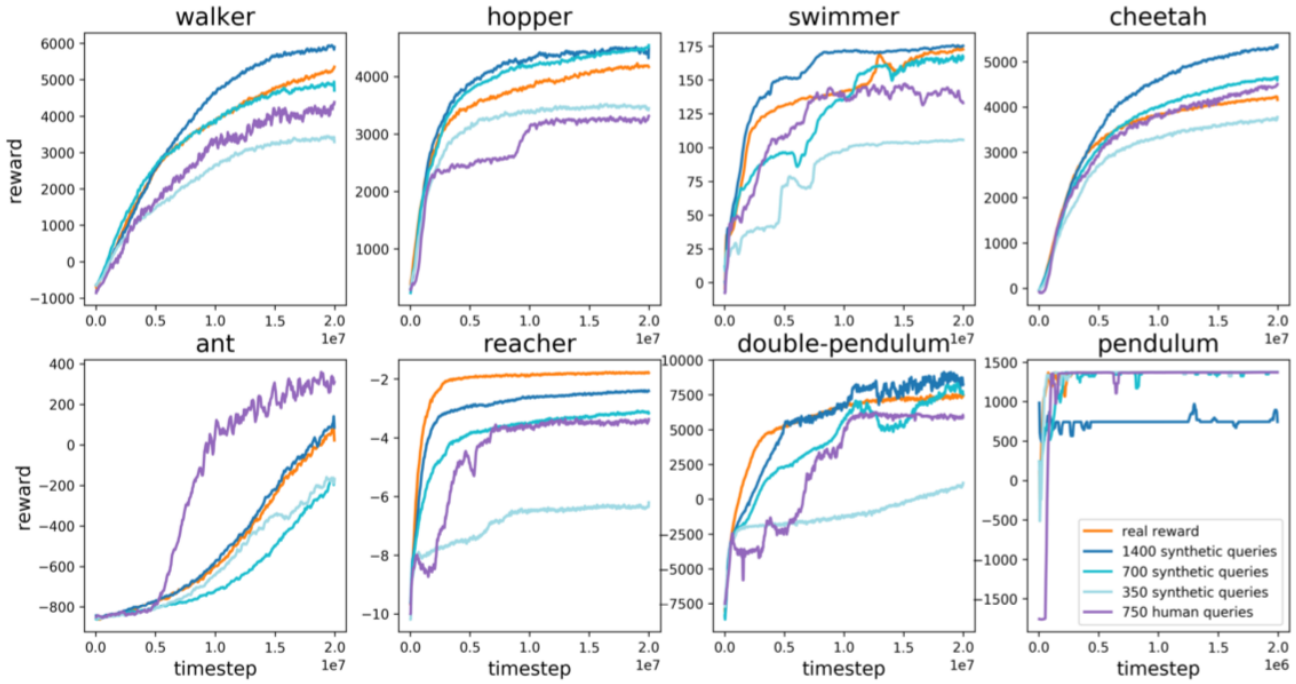


Figure 2: Results on MuJoCo simulated robotics as measured on the tasks' true reward. We compare our method using real human feedback (purple), our method using synthetic feedback provided by an oracle (shades of blue), and reinforcement learning using the true reward function (orange). All curves are the average of 5 runs, except for the real human feedback, which is a single run, and each point is the average reward over five consecutive batches. For Reacher and Cheetah feedback was provided by an author due to time constraints. For all other tasks, feedback was provided by contractors unfamiliar with the environments and with our algorithm. The irregular progress on Hopper is due to one contractor deviating from the typical labeling schedule.

The first tasks we consider are eight simulated robotics tasks, implemented in MuJoCo.

The reward functions in these tasks are linear functions of distances, positions and velocities, and all are a quadratic function of the features.

We included a simple cartpole task ("pendulum") for comparison, since this is representative of the complexity of tasks studied in prior work.

Experimental Results #2

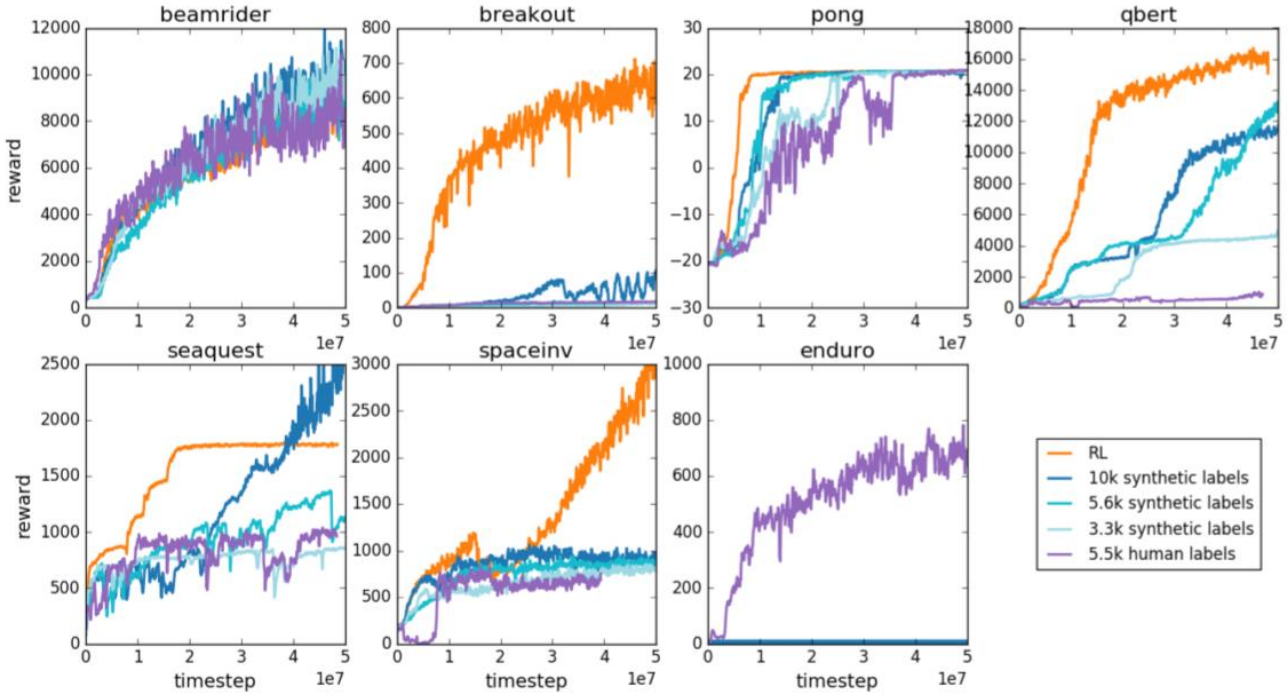


Figure 3: Results on Atari games as measured on the tasks' true reward. We compare our method using real human feedback (purple), our method using synthetic feedback provided by an oracle (shades of blue), and reinforcement learning using the true reward function (orange). All curves are the average of 3 runs, except for the real human feedback which is a single run, and each point is the average reward over about 150,000 consecutive frames.

- The second tasks is a set of seven Atari games in the Arcade Learning Environment.

Experimental Results #3

- Using the same parameters as in the previous experiments, we show that our algorithm can learn novel complex behaviors.:
- 1. The Hopper robot performing a sequence of backflips. This behavior was trained using 900 queries in less than an hour. The agent learns to consistently perform a backflip, land upright, and repeat
- 2. The Half-Cheetah robot moving forward while standing on one leg. This behavior was trained using 800 queries in under an hour.
- 3. Keeping alongside other cars in Enduro. This was trained with roughly 1,300 queries and 4 million frames of interaction with the environment; the agent learns to stay almost exactly even with other moving cars for a substantial fraction of the episode, although it gets confused by changes in background.

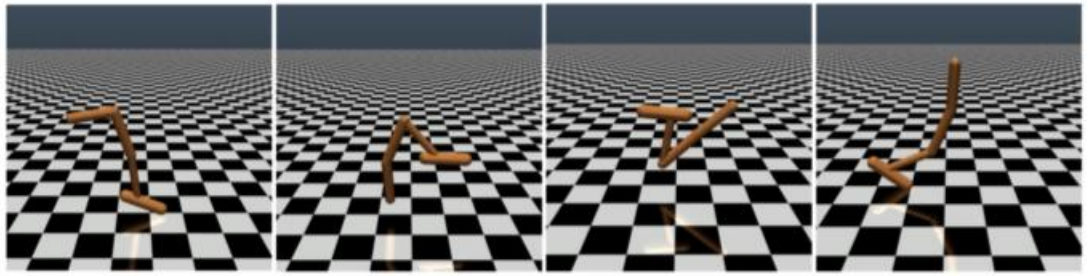


Figure 4: Four frames from a single backflip. The agent is trained to perform a sequence of backflips, landing upright each time. The video is available at this link.

Conclusion and Discussion

- Agent-environment interactions are often radically cheaper than human interaction. We show that by learning a separate reward model using supervised learning, it is possible to reduce the interaction complexity by roughly 3 orders of magnitude. Not only does this show that we can meaningfully train deep RL agents from human preferences, but also that we are already hitting diminishing returns on further sample-complexity improvements because the cost of compute is already comparable to the cost of non-expert feedback.
- Although there is a large literature on preference elicitation and reinforcement learning from unknown reward functions, we provide the first evidence that these techniques can be economically scaled up to state-of-the-art reinforcement learning systems. This represents a step towards practical applications of deep RL to complex real-world tasks.
- Future work may be able to improve the efficiency of learning from human preferences, and expand the range of tasks to which it can be applied.
- In the long run it would be desirable to make learning a task from human preferences no more difficult than learning it from a programmatic reward signal, ensuring that powerful RL systems can be applied in the service of complex human values rather than low-complexity goals.

Check Your Facts and Try Again: Improving Large Language Models with External Knowledge and Automated Feedback*

Baolin Peng† **Michel Galley**† **Pengcheng He**† **Hao Cheng**† **Yujia Xie**†
Yu Hu† **Qiuyuan Huang**† **Lars Liden**† **Zhou Yu**‡ **Weizhu Chen**† **Jianfeng Gao**†
† Microsoft Research ‡ Columbia University

Problems

- Large Language models (LLMs) have demonstrated an outstanding ability in generating fluent, coherent, and informative natural language texts. It is commonly understood that the impressive capabilities of these models stem from the abundance of world knowledge encoded therein and models' ability to generalize from that knowledge.
- However, the knowledge encoding of LLMs is lossy and the knowledge generalization could lead to "memory distortion." As a result, these models tend to *hallucinate*.
- Constant changes in real-world settings cause LLMs to quickly become stale for time-sensitive tasks.
- While there is a growing interest in improving LLMs using external knowledge almost all the previously proposed methods require finetuning the parameters of a LLM, which can be prohibitively expensive as the size of LLMs grows exponentially.

Method: LLM-Augmentor

- Improve LLMs with external knowledge and automated feedback using PnP (plug-and-play) modules: Working Memory, Policy, Action Executor, and Utility.
- We formulate human-system conversation as a Markov Decision Process (MDP) described by a five-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$:

- $\mathcal{S}$ is an infinite set of dialog states, which encode information stored in Working Memory, including dialog history, user query, evidence, candidate response;
- $\mathcal{A}$ is a set of actions that Policy picks to execute, including (1) calling Knowledge Consolidator to consolidate evidence from external knowledge and (2) calling Prompt Engine to query the LLM to generate candidate responses;
- $P(s'|s, a)$ gives the transition probability of entering a new state s' after action a is taken in state s;
- $R(s, a)$ is the external reward received after taking action a in state s, which is provided by the environment (*e.g.,* users or simulators); and
- $\gamma \in (0, 1]$ is a discount factor.

Method: LLM-Augmenter

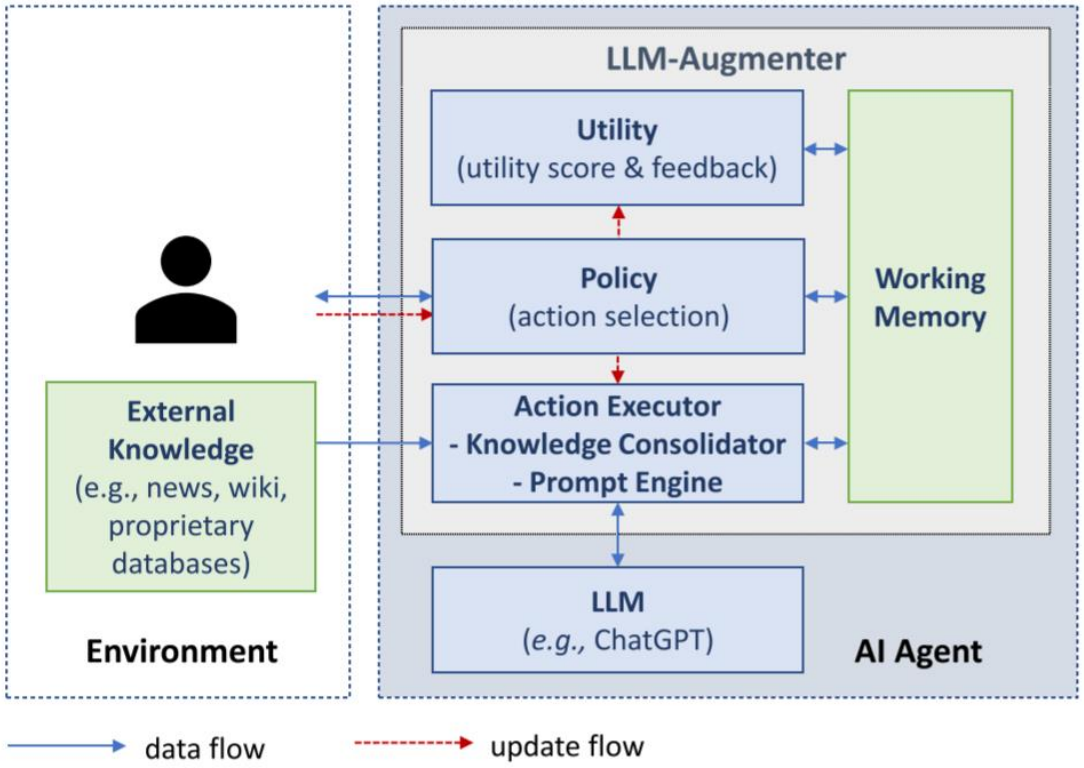


Figure 2: LLM-AUGMENTER architecture showing how its plug-and-play modules interact with the LLM and the user's environment.

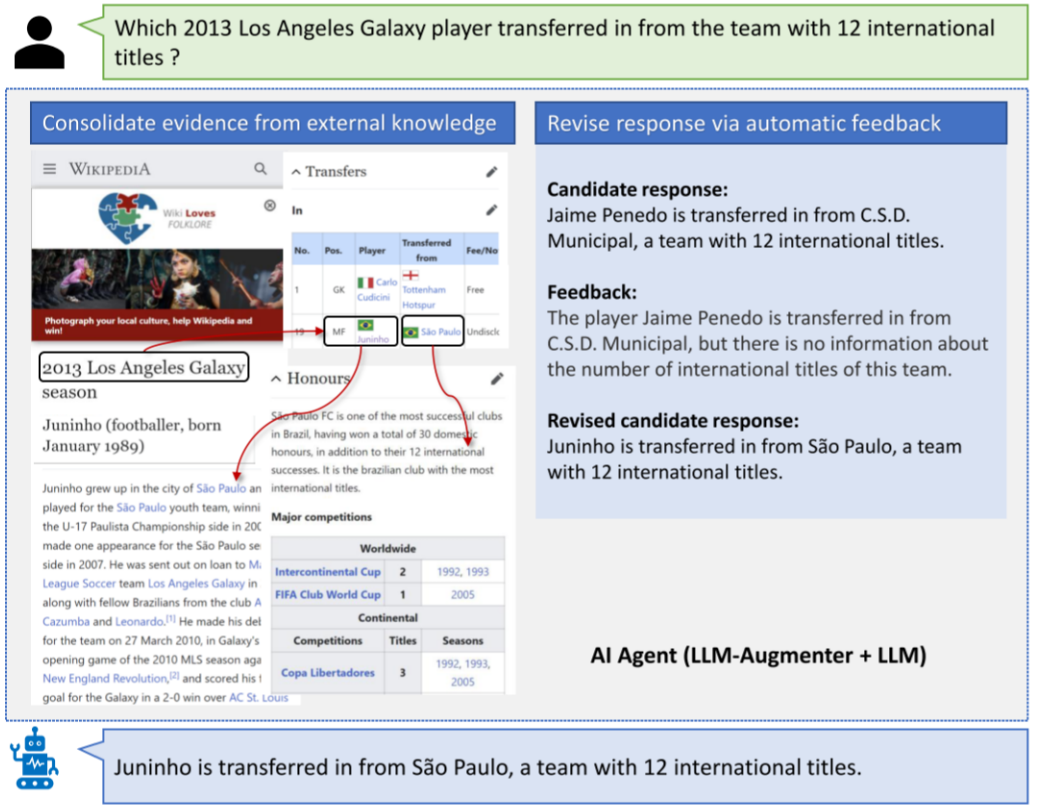


Figure 1: LLM-AUGMENTER improves a fixed LLM by (1) consolidating evidence from external knowledge for the LLM to generate responses grounded in evidence, and (2) revising LLM's (candidate) responses using automated feedback.

Working Memory

- This module tracks the dialog state that captures all essential information in the conversation so far. The state is represented using a six-tuple (q, e, o, u, f, h_q):

- q is the current user query;
- e is evidence for q, consolidated from external knowledge by Knowledge Consolidator;
- o is a set of the LLM-generated candidate responses for q;
- u is a score assessing the utility of each element of o, and f is a verbalized feedback to guide the LLM to improve its utility — both u and f are generated by the Utility module (see Section 2.4); and
- h_q is the dialog history before q.

- Note that given user query *q*, LLM-Augmenter can take multiple iterations to revise its response, with each iteration generating a candidate response based on evidence, feedback and utility, before sending the final response to the user.

Policy

- This module selects the next system action that leads to the best expected reward R. These actions include (1) acquiring evidence e for q from external knowledge, (2) calling the LLM to generate a candidate response, and (3) sending a response to users if it passes the verification by the Utility module.
- In this study, we implement a trainable policy π as a neural network model parameterized by θ.

$$\underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{s\sim\mathcal{S},a\sim\pi_\theta}[R(s,a)] \qquad (1)$$

Policy

- Policy learning typically requires large amounts of human-machine interactions, which can be costly to collect. To address the challenge, policy learning can be done in three stages:

- **Bootstrapping from a rule-based policy**: Domain experts encode task-specific knowledge and business logic into IF-THEN rules. For example, if a product name is mentioned in a user query for customer service, it is wise to always call Knowledge Consolidator to collect information of the product from a product database.
- **Learning with user simulators**: We use a *language model to simulate how human users interact with LLM-Augmenter*. Any valid response from LLM-Augmenter that passes the evaluation of the Utility module can be used as a training example, allowing LLM-Augmenter to self-improve.
- Finally, LLM-Augmenter interacts with human users to further refine its policy.

Action Executor

- It is composed of two components, ***Knowledge Consolidator*** and ***Prompt Engine***.
- The Knowledge Consolidator augments LLMs with the capability of grounding their responses on external knowledge to mitigate hallucination when completing tasks. The Knowledge Consolidator is designed in a modular fashion, consisting of a *knowledge retriever*, an *entity linker* and, an *evidence chainer*.
- The retriever first generates a set of search queries based on q and h_q, and then calls a set of APIs to retrieve raw evidence from various external knowledge sources.
- The raw evidence is sometimes incomplete and noisy. Thus, the entity linker enriches raw evidence with related context to form evidence graphs, i.e., linking each entity mentioned in raw evidence to its corresponding description based on Wikipedia.
- Then, the chainer prunes irrelevant evidence from the graphs and forms a shortlist of evidence chains that are most relevant to queries. The consolidated evidence e is then sent to Working Memory.

Action Executor

- The Prompt Engine generates a prompt to query the LLM to generate a (candidate) response o for q.
- The prompt is a text string that consists of task instruction, user query q, dialog history h_q, evidence e if it is made available by Knowledge Consolidator, and feedback f if it is made available by the Utility module.

Utility

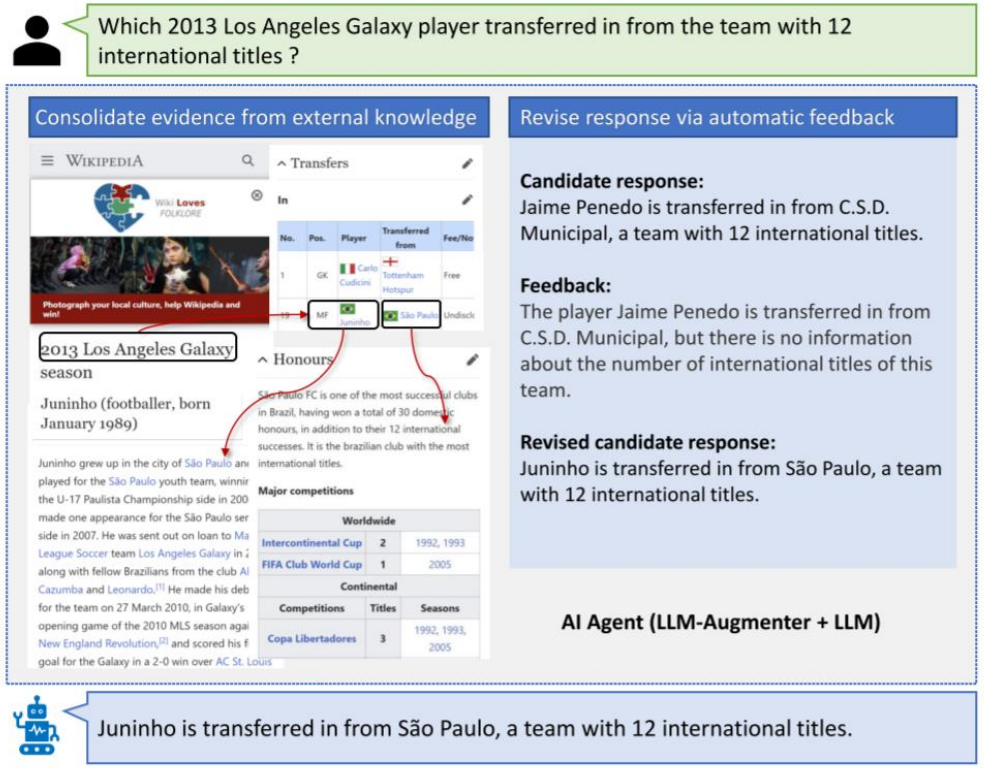
- Given a candidate response o, the Utility module generates utility score u and a corresponding feedback f using a set of task-specific utility functions.
- These utility functions access the alignment of the LLM's responses with user expectations or specific business requirements.
- There can be two distinct types of utility functions:

- *Model-based utility functions* assign preference scores to different dimensions of a response, such as fluency, informativeness and factuality. These functions are trained on pre-collected human preference data or annotated log data.
- *Rule-based utility functions*, implemented using heuristics or programmed functions, measure whether a response complies with a specific rule.

Utility

- In addition, we have developed a utility function to generate informative and actionable feedback to help revise prompts to allow the LLM to generate better responses.
- As shown in Figure 1, the utility function generates feedback "but there is no information about the number of international titles." Such a utility function is a text generation model Q parameterized by ψ, and can be implemented as a seq2seq or auto-regression language model. It tasks as input user query q, evidence e, candidate response o and dialog history h_q, and generates feedback in text f as

$$f = Q_\psi(q, e, o, h_q) \qquad (2)$$



Information Seeking Dialog

- **Datasets**: DSTC7 Track 2 for News Chat, Customer Service DSTC11 Track 5 for Customer Service.
- **Language Model**: ChatGPT
- **Knowledge Consolidator**: For News Chat: a BM25 retriever over web documents linked from Reddit posts. For the Customer Service task: a BM25-based retriever over the knowledge bases of FAQs and Yelp reviews.
- **Utility**: We use the utility score, *Knowledge F1* to measure the overlap between a prediction and evidence which is either consolidated by Knowledge Consolidator or provided as golden knowledge. Feedback generation is accomplished using a template-based natural language generator. For example: If the KF1 score falls below a certain threshold, the feedback is "The response is inconsistent with the knowledge. Please generate again."
- **Policy**: we use a rule-based policy for our experiments involving ChatGPT. Additionally, to test the viability of LLM-Augmenter with a trainable policy, we employ offline RL to train the parameters of Policy as Equation 1, where the policy model is based on T5-Base.

Information Seeking Dialog

- **Prompt Engine:**

I want you to act as a chatbot. You need to answer user' questions nicely. Context: User: $[U_1]$ Assistant: $[R_1]$ ... User: $[U_t]$ Assistant: $[R_t]$
I want you to act as a chatbot. You will be presented with knowledge snippets. You need to answer user' questions nicely and accurately based on the knowledge snippets. Working Memory: $[M_t]$ Context: User: $[U_1]$ Assistant: $[R_1]$ ... User: $[U_t]$ Assistant: $[R_t]$

Table 7: Prompt Templates for News Chat. LLMs generated responses is highlighted with [ ].

I want you to act as a chatbot AI for travel planning. You need to answer customer's questions nicely. Context: User: $[U_1]$ Assistant: $[R_1]$ ... User: $[U_t]$ Assistant: $[R_t]$
I want you to act as a chatbot AI for travel planning. You will be presented with knowledge snippets. You need to answer customer's questions nicely and accurately based on the knowledge snippets. Working Memory: $[M_t]$ Context: User: $[U_1]$ Assistant: $[R_1]$ ... User: $[U_t]$ Assistant: $[R_t]$

Table 8: Prompt Templates for Customer Service. LLMs generated response are highlighted with [ ].

Automatic Evaluation Results:

- The impact of using external knowledge
- The impact of using automated feedback

Model	K.C.	Feedback	KF1 ↑	BLEU ↑	ROUGE ↑	chrF ↑	METEOR ↑	BERTScore ↑	BARTScore ↑	BLEURT ↑	Avg. length
ChatGPT	-	-	26.71	1.01	16.78	23.80	7.34	82.14	0.25	26.98	58.94
LLM-Augmenter	BM25	✗	34.96	6.71	22.25	27.02	9.35	**83.46**	0.34	26.89	46.74
LLM-Augmenter	BM25	✓	**36.41**	**7.63**	**22.80**	**28.66**	**10.17**	83.33	**0.35**	**27.71**	54.24
LLM-Augmenter	gold	✗	*57.44*	*19.24*	*38.89*	*40.02*	*17.21*	*86.65*	*0.82*	*40.55*	*44.35*
LLM-Augmenter	gold	✓	*60.76*	*21.49*	*40.56*	*42.14*	*18.50*	*86.89*	*0.93*	*42.15*	*47.19*

Table 1: Evaluation scores (in %) and average response lengths for the News Chat (DSTC7) dataset. BM25: Each model retrieves 5 knowledge snippets from the corresponding knowledge source. K.C. denotes Knowledge Consolidator.

Model	K.C.	Feedback	KF1 ↑	BLEU ↑	ROUGE ↑	chrF ↑	METEOR ↑	BERTScore ↑	BARTScore ↑	BLEURT ↑	Avg. length
ChatGPT	-	-	31.33	4.70	24.02	27.14	12.83	87.88	1.53	**47.99**	28.81
LLM-Augmenter	BM25	✗	34.07	**4.78**	**24.52**	28.95	13.61	**87.96**	1.78	47.21	32.65
LLM-Augmenter	BM25	✓	**37.41**	3.86	24.20	**30.90**	**14.74**	87.58	**2.09**	44.71	45.07
LLM-Augmenter	gold	✗	*45.63*	*6.54*	*29.77*	*33.32*	*16.93*	*89.35*	*2.59*	*54.38*	*33.04*
LLM-Augmenter	gold	✓	*52.83*	*5.63*	*29.65*	*35.68*	*18.66*	*89.01*	*3.14*	*52.49*	*45.09*

Table 2: Evaluation scores (in %) and average response lengths for the Customer Service (DSTC11) dataset. BM25: Each model retrieves 5 knowledge snippets from the corresponding knowledge source. K.C. denotes Knowledge Consolidator.

Automatic Evaluation Results:

- The impact of using trainable Policy

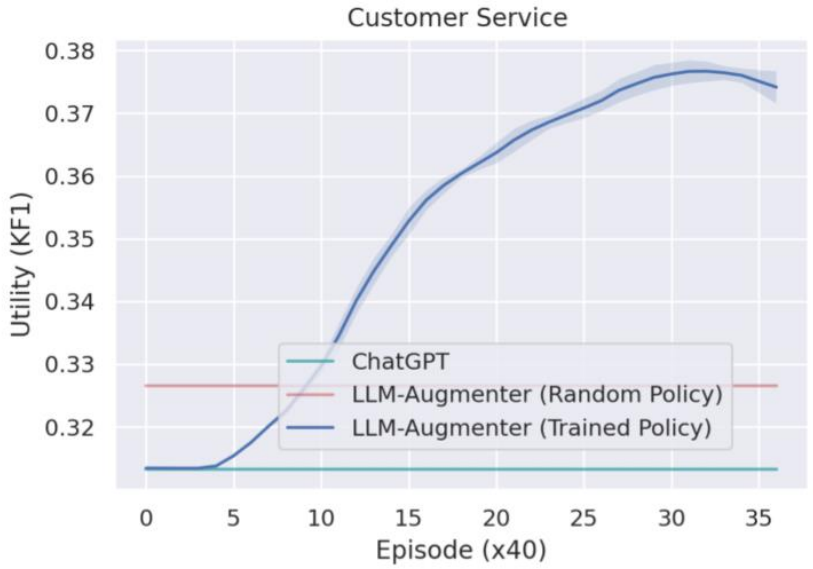


Figure 3: Learning curve of ChatGPT-Companion with T5-Base as the policy module. The solid curves are the mean and the shaded regions are the maximum and minimum utility scores over 5 runs.

Human Evaluation Results:

Model	Usefulness ↑	Humanness ↑
ChatGPT	34.07	30.92
LLM-AUGMENTER	**45.07**	**35.22**

Table 3: Human evaluation of ChatGPT and LLM-AUGMENTER with BM25 in Customer Service scenario. All differences are significant ($p < 0.05$). Inter-annotator agreements according Krippendorff's alpha (interval metric) are 0.15 and 0.07 respectively.

Wiki QA

- **Datasets**: OTT-QA
- **Knowledge Consolidator**: Wikipedia passages and tables as the knowledge source. Instead of using BM25 as done for dialog tasks, we resort to a dense model, DPR, as the backbone retriever.
- **Utility**: We use recall as the utility score, i.e., preferring responses with higher token overlap with the corresponding evidence set.
- **Prompt Engine:**

I am a highly intelligent question answering bot that can answer questions. If you ask me a question that is rooted in truth, I will give you the answer. If you ask me a question that is nonsense, trickery, or has no clear answer, I will respond with "Unknown".
Question: $[Q]$
Answer: $[A]$

I am a highly intelligent question answering bot, and can answer questions given some documents and tables. If you ask me a question that is rooted in truth, I will give you the answer. If you ask me a question that is nonsense, trickery, or has no clear answer, I will respond with "Unknown".
Working Memory: $[M]$
Question: $[Q]$
Answer: $[A]$

Results

Model	Knowledge Consolidator	Feedback	Wiki QA		
			P ↑	R ↑	F1 ↑
CHATGPT	-	-	0.48	1.52	0.59
LLM-AUGMENTER	DPR	✗	2.08	4.31	2.38
LLM-AUGMENTER	CORE	✗	7.06	14.77	8.08
LLM-AUGMENTER	CORE	✓	8.93	33.87	11.80

Table 5: Evaluation results on Wiki QA. Each model retrieves top-5 knowledge snippets from the corresponding knowledge source. The top-5 answer recall of consolidated evidence (CORE) is 50.83.

THANK YOU!