Adapting LLM Agents with Universal Communication Feedback

Kuan Wang [1] **Yadong Lu** [2] **Michael Santacroce** [2] **Yeyun Gong** [3] **Chao Zhang** [1] **Yelong Shen** [2]

Abstract

Recent advances in large language models (LLMs) have demonstrated potential for LLM agents. To facilitate the training for these agents with both linguistic feedback and non-linguistic reward signals, we introduce Learning through Communication (LTC). We design a universal buffer to store all the feedback, and an iterative pipeline to enable an LLM agent to explore and update its policy in an given environment. To utilize our universal buffer for capturing agent interactions in various tasks, we introduce diverse communication patterns tailored for both single-agent and multi-agent environments. We evaluate the effectiveness of our LTC approach on four diverse datasets: ALFWorld (single-agent), HotpotQA (multi-agent collaboration), Chameleon (multi-agent competition), and GSM8k (multi-agent teacher-student). On these datasets, LTC outperforms supervised instruction fine-tuning baselines by 3.6% to 12%. These results demonstrate the versatility and effectiveness of LTC in facilitating online adaptation for LLM agents.

1. Introduction

Recent advances in large language models (LLMs) (Ouyang et al., 2022; Bubeck et al., 2023; Wei et al., 2022a) have shed light on human-like LLM agents. In addition to design prompting methods (Wei et al., 2022b; Yao et al., 2023; Wu et al., 2023a), recent works also focus on how to train the LLMs agent use linguistic feedback and non-linguistic reward signals. The linguistic feedback is usually processed as instruction data to do Instruction Fine-tuning (IFT) (Chung et al., 2022; Lee et al., 2023; Honovich et al., 2022; Wang et al., 2022e), while the non-linguistic reward signals are generally used to do alignment with human preference (Ouyang et al., 2022; Bai et al., 2022a; Stiennon et al., 2020; Leike et al., 2018).

While some scenarios provide agents with heterogeneous feedback, existing methods can only utilize the feedback partially. For instance, in multiplayer board role-playing games, players generate a wealth of linguistic data, and the game concludes with definitive reward signals indicating victory or defeat. Current approaches employ the linguistic data for IFT (Li et al., 2023; Micheli & Fleuret, 2021), while the reward signals serve solely as a filtering criterion to select the IFT data instead of the objective of reinforcement learning.

To address this gap, we propose a universal framework, named Learning through Communication (LTC), to train LLM agents with both linguistic feedback and non-linguistic reward signals. We design a universal buffer to store all the feedback, and an iterative pipeline to enable an LLM agent to explore and update its policy in an given environment. Each iteration of LTC comprises two distinct phases: (1) ***Exploration***: During this phase, the agent interacts with the environments and other agents to gather diverse trajectories (linguistic) and reward signals (non-linguistic) into the universal buffer. (2) ***Updating***: In this phase, the agent's model is updated based on the collected data in the universal buffer. For updating, LTC combines the language modeling loss and the PPO loss to strike a balance between language consistency and reward signals As the pivot of the iterative pipeline, the replay buffer is updated after each exploration phase, and a subset of the buffer is sampled for the updating phase.

To universally supports linguistic feedback and non-linguistic reward signals during communication, we design the replay buffer structure as a trajectory of tokens sequences (Figure 3). Such a replay buffer structure is applicable to diverse tasks, including single-agent and multi-agent environments. To facilitate collecting trajectories with linguistic data and reward signals, we devised three communication patterns: (1) ***Single-agent Monologue***: This pattern allows a single agent to collect trajectories contain linguistic data and receive reward signals from the environments. (2) ***Multi-agent Dialogue***: This pattern enables multiple agents to interact with each other and external tools to collect linguistic data, and utilize reward signals provided

[1]Georgia Institute of Technology [2]Microsoft Azure AI [3]Microsoft Research. Correspondence to: Kuan Wang <kuanwang@gatech.edu>, Yelong Shen <yeshe@microsoft.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

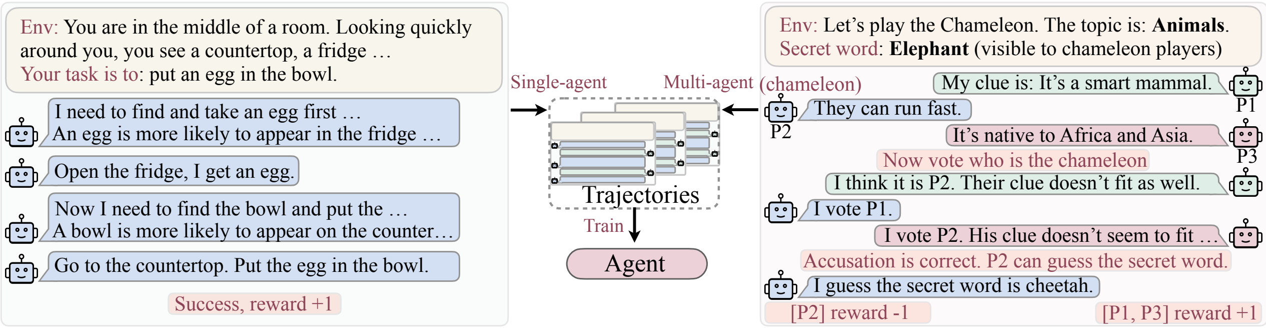

Figure 1. The LTC framework is adept for both single-agent and multi-agent environments. Within these environments, agents have the capability to persistently engage in exploration and interaction to collect trajectories through various communication patterns. Concurrently, LTC facilitates the training of these agents utilizing the data acquired from their exploratory activities. This process enables the agents to autonomously adapt to their respective environments, negating the necessity for human supervision.

by the environments. (3) ***Teacher-student Dialogue***: This variant of multi-agent dialogue that collect the linguistic feedback and non-linguistic reward signals provided by a teacher agent instead of the environment.

We evaluate LTC method on several representative datasets: *ALFWorld* for decision-making, *HotpotQA* for knowledge-intensive reasoning, and *GSM8k* for numerical reasoning. Throughout these experiments, LTC consistently outperforms the baselines. In *ALFWorld*, LTC outperforms the strong instruction tuning baseline by 12% on success rate, even in the challenging Pick 2 task. This shows that our communication mechanism enables the agent to learn from its experiences for task solving. On *HotpotQA*, LTC outperforms the instruction tuning baseline by 5% on EM score, and our Llama-7B based agent even obtains slightly better (0.6%) performance than the ReAct-Tuning baseline which uses 9× larger PaLM-62B model. On *GSM8k*, LTC also beats the CoT-Tuning baseline by 3.6% on accuracy. These results highlight the adaptability and effectiveness of LTC approach across varied domains.

Our key contributions are:

1. **Learning through Communication (LTC)**: We propose a universal framework, named Learning through Communication (LTC), to train LLM agents with **both linguistic feedback and non-linguistic reward signals**. We design a universal buffer to store all the feedback, and an iterative pipeline to enable an LLM agent to explore and update its policy in an given environment.
2. **Task-specific Communication Patterns**: The LTC paradigm allows for flexible design of communication patterns tailored to different tasks. We introduce three specific patterns: Single-agent Monologue, Multi-agent Dialogue, and Teacher-student Dialogue. These patterns can be combined to generate diverse structured interactions and feedback signals for agent training, catering to various task types.
3. **Empirical Study and Findings**: We conduct rigorous experiments on public benchmark tasks to demonstrate the effectiveness of LTC. Our results indicate that LTC can be a superior approach compared to instruction-tuning or prompting baselines.

2. Learning Through Communication

We design Learning Through Communication (LTC), an iterative training method for LLM agents to continuously adapt to new environments. As shown in Figure 2, LTC iterates between two phases: (1) An exploration phase where agents can interact with new environments and other agents to collect trial data with feedback, and (2) a updating phrase to fine-tune the agent to update the policy.

2.1. Exploration Phase

At the start of each iteration, the agent explores the environments to get the trajectories and the reward signal data. We denote these data as a tuple: $\mathcal{S} = (\mathcal{T}, \mathcal{M}, \mathcal{R})$, where $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ represents the text data generated by the communication process during agent exploration, $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ with $m_i \in \{0, 1, 2\}$ indicates the source of the text data (system or agents), $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ with $r_i \in \{-1, 0, 1\}$ represents the reward signals provided by either the system or the agents. We demonstrate the the details of this data structure in Figure 3, $\mathcal{M}$ is the mask list, and $\mathcal{R}$ is the reward list. In PPO training, both the value list and the log-prob list correspond directly to the action list. For brevity, we denote these three lists together as $\mathcal{T}$ here. Please see Appendix C.3 for more detaills.

To collect the trajectories data $\mathcal{S} = (\mathcal{T}, \mathcal{M}, \mathcal{R})$ from different types of tasks, we design the communication patterns for these tasks. Here we provide three communication patterns:

- **Single-agent Monologue**: Single-agent Monologue is a single-agent soliloquy style communication pattern, designed for general instruction following tasks (Algorithm 1). It split the tasks into step by step like ReAct and CoT, and their own trajectories with system rewards are collected to train themselves at the same time with their exploration. Figure 1 left is a toy example of *ALFWorld* to demonstrate the Monologue pattern with a single agent. This agent soliloquizes to think the situation and take the actions to explore the environment and finally get the reward provided by the environment. This pattern is based on the think and act steps in the ReAct formulation (Yao et al., 2023), we design the training buffer collection process to make it aligh with our reinforcement learning formulation.
- **Multi-agent Dialogue**: Multi-agent Dialogue is a multi-agent discussion style pattern (Algorithm 2). It is designed for multi-agent collaborating and competing tasks, where multiple agents will play their role by speaking or taking actions in a certain order and **a final reward will be given by the environment** based on the performance of the agents. The left figure of Figure 4 is a toy example of *HotpotQA* to illustrate this pattern for collaborating, where the GPT-4 agent play as a thinker to analyze the situation and give suggestions to the actor agent who is responsible for making decisions. The reward in *HotpotQA* is the correctness of the answer obtained by two agents. And we can use their communication data to train the LTC agents do both thinker and actor so that they can learn how to cooperate with each other to solve the task. The right figure of Figure 1 is a toy example of Multi-agent Dialogue for a competing game task Chameleon, where three agents play different roles. The reward is the win or loss of the game, so they need do with deduction and bluffing in the communication process to win the game. And their games trajectories will be use in LTC iterations to boost the agents.
- **Teacher-student Dialogue**: Teacher-student Dialogue is a teacher-student style pattern for powerful agents to teach the novice agents (Algorithm 3). We design this pattern for complex analytical tasks such as numerical reasoning, which require extensive analytical examples for agents to improve the specific reasoning ability lacking in the pretrained models. Teacher-student Dialogue pattern has two roles (student and teacher) played by two agents, however, in addition to the linguistic feedback, the teacher roles can **directly provide the non-linguistic reward signals**, which are all provided by the system (environments) in the previous pattern. The right figure of Figure 4 is a toy example with GSM8k to demonstrate how the student agent communicates with the teacher agent in a homework-correcting style. In the math question environment, the student agent starts with an initial answer to the current question, then the teacher directly corrects the answer with a reward. To help the student improve ability instead of just memorizing the solution, the teacher will generate another individual question and provide a new reward to the student.

2.2. Updating phase

In the updating phase, the LLM agent model could be optimized through the conversation sessions collected in the exploration stage. Given a example session $\mathcal{S} = (\mathcal{T}, \mathcal{M}, \mathcal{R})$, we mainly utilize two training objects for model training.

- Language model Objective: $\mathcal{L}_{\text{LM}}$ encourages the model to learn from the trajectory $\mathcal{T}$, serving as an unsupervised learning schema to help model for behavior cloning from other agents' response or predicting system feedbacks.
- Reinforcement Objective: $\mathcal{L}_{\text{reinforce}}$ optimizes the model by maximizing the expectation reward provided by environment or a teacher agent (i.e., GPT-4 (OpenAI, 2023)). It is an goal-oriented objective, and allows the model to learn through both positive and negative signals in the communication session.

Thus, the overall training objective for LTC combines the above two terms:

$$\mathcal{L}_{\text{LTC}}(\mathcal{S}) = \beta\mathcal{L}_{\text{LM}}(\mathcal{T}) + \mathcal{L}_{\text{reinforce}}(\mathcal{S}), \tag{1}$$

where β is a balancing hyper-parameter. The off-policy PPO algorithm (Schulman et al., 2017) is utilized for optimizing $\mathcal{L}_{\text{reinforce}}(\mathcal{S})$, and it can be further breakdown into policy loss, value loss and policy entropy regularization terms in implementation. The vanilla PPO algorithm takes the triplet $(\text{state}, \text{action}, \text{rewards})$ for training. In this case, we sample from the trajectories $(\mathcal{T}_{<i}, t_i)$ for simulating the state-action pairs, specifically, we only keep the tokens generated by agent model itself as actions for policy updating.

3. Experiments

3.1. Settings

Model Architecture We use a modified version of Llama (Touvron et al., 2023) as the base model. To generate state values corresponding to the action tokens, we introduce an additional linear layer to serve ast the value head. This value head acts as an auxiliary output module, and the output values are processed using the $tanh()$ function to ensure they fall within the range of (-1, 1). This adaptation

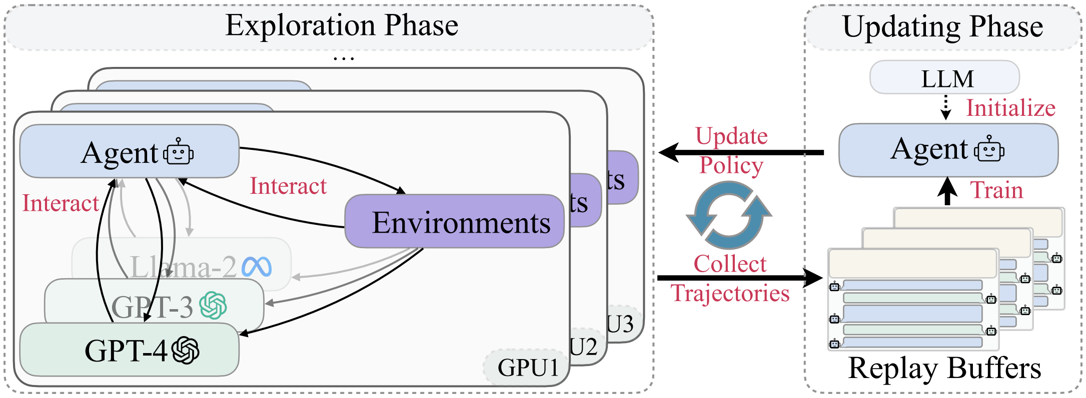

Figure 2. LTC has an iterative two-phase framework. During the exploration phase, the agent proactively explores new environments and communicates with other agents, gathering the trajectories to update the replay buffer. Then the agent is trained for updating the policy in the updating phase.

for RL has also been discussed in prior studies (Santacroce et al., 2023).

Agent Pre-training We use the Llama-7B model (Touvron et al., 2023) for our LLM agent. To enhance the agent's ability to follow task-specific instructions, we initialize it by instruction fine-tuning (IT). And this initialized agent works as the baseline for a fair comparison. This step is crucial because the original Llama-7B model, without prior instruction fine-tuning, struggled to follow task instructions and generation sensible actions in the environments. To collect data for instruction fine-tuning, we employ GPT3/4 as our agent to explore the environments created from the training set. We then filter out negative examples and retain positive examples to train the initial agent. For both the *ALFWorld* and *HotpotQA* datasets, we leverage GPT3 (specifically, text-davinci-003). However, for the *GSM8k* dataset, we use GPT4 due to GPT3's inadequate performance in handling mathematical problems, which resulted in a scarcity of positive examples.

Training details We utilize the AdamW optimizer (Loshchilov & Hutter, 2017) with a batch size of 32. The learning rate is set to 2e-4. In each iteration, the sizes of new environments for agents to explore are: 256 for *ALFWorld*, 512 for *GSM8k*, and 1024 for *HotpotQA*. For parameter-efficient fine-tuning, we employ LoRA (Hu et al., 2021) with hyperparameters $R = 16$ and $\alpha = 16$. For distributed training, we utilize 4 nodes with 8×A100 GPUs on *HotpotQA* and *GSM8k*. For the experiments on *ALFWorld*, we use 1 node with 2×A100 GPUs due to the dataset's small scale.

Baselines We compare the agents trained by LTC with existing prompting and instruction tuning methods, including ReAct (Yao et al., 2023), ReAct-IM (Huang et al., 2022b), CoT (Wei et al., 2022b), CoT-SC (Wang et al., 2022c;d), BUTLER (Micheli & Fleuret, 2021). The detailed of these baselines are described in Appendix C.8. Most of these

Method \ Task	Pick	Clean	Heat	Cool	Look	Pick 2	All
ReAct (avg)	65	39	83	76	55	24	57
ReAct (best of 6)	**92**	58	**96**	86	78	41	71
ReAct-IM (avg)	55	59	60	55	23	24	48
ReAct-IM (best of 6)	62	68	87	57	39	33	53
BUTLER$_g$ (best of 8)	33	26	70	76	17	12	22
BUTLER (best of 8)	46	39	74	**100**	22	24	37
ReAct-Tuning (avg)	83	91	91	90	72	8	77
ReAct-Tuning (best of 3)	**92**	**97**	**96**	95	78	24	78
LTC (avg)	89	91	93	97	96	67	90
LTC (best of 3)	**92**	**97**	**96**	**100**	**100**	**76**	**91**

Table 1. AlfWorld success rates (%) for 6 tasks. The results of the bottom block are obtained by fine-tuning Llama-7B model.

methods focus on few-shot prompting, and different pre-trained models are used. To ensure a fair comparison, we include the additional baselines named ReAct-Tuning and CoT-Tuning by fine-tuning the Llama-7B model using the collected trajectories as fine-tuning data. In addition, GPT-4 are not used in the test time, and all the results reported are obtained by the trained agent itself.

3.2. Results

ALFWorld As shown in Table 1, LTC outperforms the previous best methods* on all of tasks of *ALFWorld*. We can see that Instruction Fine-tuning is already a strong baseline outperforming others, yet our LTC achieves a success rate of 91%, remarkably outperforming the best Instruction Tuning baseline (78%). Notably, on both Cool and Look tasks, LTC obtains a 100% success rate. Even on the hardest Pick Two & Place task (e.g., "put two pencils in the drawer"), it achieves a decent 76% success rate. The Pick Two task requires the agent to perform two sequences of "pick and place" actions in one task, while keeping track of the desired type and the location. The combined sequences and the need

*For *ALFWorld*, ReAct and ReAct-IM results are from Table 3 of (Yao et al., 2023). BUTLER and BUTLER$_g$ results are from Table 4 of (Shridhar et al., 2020b), and they are trained with DAgger (Ross et al., 2011).

to remember the previous location make this task challenging. This may be the reason why baselines achieve lower success rates on this task. In contrast, our LTC agent, which further trains the agent with self-exploration significantly outperforms other agents. This underscores the effectiveness of the communication mechanism in LTC.

Model	Method	EM score
PaLM-540B	CoT (Wei et al., 2022b)	29.4
	CoT-SC (Wang et al., 2022c)	33.4
	ReAct (Yao et al., 2023)	27.4
	ReAct → CoT-SC	35.1
GPT3-175B	ReAct	30.8
PaLM-62B	ReAct-Tuning	32.6
	CoT-Tuning	25.2
PaLM-8B	ReAct-Tuning	25.0
	CoT-Tuning	14.0
Llama-7B	ReAct-Tuning	28.2
	LTC(single-agent monologue)	31.0
	LTC(multi-agent dialogue)	**33.2**
Llama2-13B	ReAct-Tuning	33.8
	LTC(multi-agent dialogue)	**35.8**

Table 2. EM scores on HotpotQA with prompt and tuning methods. Methods that use fine-tuning are marked by "-Tuning".

HotpotQA As shown in Table 2, LTC outperforms the instruction tuning baseline[†] by 5% on Exact Match (EM) score, and it even outperforms ReAct and CoT on their default settings. Note that ReAct and CoT use PaLM-540B and GPT3-175B as the pre-trained LM model, which is 77x and 25x larger than our the Llama-7B model we used. By sampling 21 CoT trajectories during inference and adopting the majority answer, CoT-SC is slightly better (0.2%) than LTC, and their combined method ReAct → CoT-SC surpasses LTC by 1.9%. Compared to other models with tuning, our Llama-7B based agent even obtains slightly better (0.6%) performance than the ReAct-Tuning baseline with 9× larger PaLM-62B model.

Chameleon As shown in Table 3, LTC outperforms the instruction tuning baselines by 3.1% on winning rate against GPT-4 players. In the training, all the players are played by the same Llama2-7B model that we are training. While in the testing, to get the winning rate of our trained agent against GPT4, only 1 player is randomly picked to use our trained agent as backend, and other players are played by GPT4. We could see that the LTC agents winning rate improves with the increasing of number of players, we explain this by the more players, the higher chance that the GPT4 players carry the game.

[†]For HotPotQA, Prompting method results without fine-tuning are from Table 1&5 of (Yao et al., 2023). PaLM-8B and PaLM-62B scores are estimates from Figure 3 of (Yao et al., 2023).

Method \ #players	n=3	n=4	n=5	overall
Llama-Tuning	20.8	20.3	23.8	21.9
Llama-LTC	**22.9**	**23.4**	**27.5**	**25.0**

Table 3. Chameleon game winning rates (%) of different number of players settings. At each game, one player is played by target evaluated model, and the others are played GPT-4.

GSM8k As shown in Table 4, LTC(teacher-student dialogue) outperforms the instruction fine-tuning baseline by 3.6% on accuracy, and it surpasses the LTC(single-agent monologue) baseline, which does not use the reward and feedback from GPT-4.

Model	Method	Accuracy
PaLM-540B	CoT (Wei et al., 2022b)	56.5
	CoT-SC (Wang et al., 2022c)	74.4
GPT3-175B	CoT (Wei et al., 2022b)	60.1
	CoT-SC (Wang et al., 2022c)	78.0
Llama-7B	CoT (Touvron et al., 2023)	11.0
	CoT-SC (Touvron et al., 2023)	18.1
Llama-7B	CoT-Tuning	37.7
	LTC(single-agent monologue)	39.6
	LTC(teacher-student dialogue)	**41.3**

Table 4. Accuracy on GSM8k. The results of the bottom block are obtained by fine-tuning LLaMA-7B model, while the others are prompting methods without fine-tuning.

However, LTC underperforms CoT and CoT-SC with the much larger models (PaLM-540B and GPT3-175B). This phenomenon is because numerical reasoning requires a larger model size and sufficient pretraining data, as observed in (OpenAI, 2023). Unfortunately, due to computational resource limitations, we can only train the relatively small Llama-7B model but were unable to train larger-scale models. Nevertheless, we believe that exploring LTC with larger models is promising for future research.

4. Conclusion

We introduced LTC, a paradigm that adapts LLM agents to new tasks and environments via communication-based iterative learning. Within this LTC framework, we have designed three communication modes to facilitate interactions between LLM agents and their environments. The history of these interactions can be autonomously organized into training data for PPO training so that the agent can adapt to the new task. Our approach represents a closed loop where the agent self-interacts with the environment or other agents, and learning to improve itself with minimal human intervention. Empirically, LTC consistently outperforms existing LLM agent and instruction tuning baselines, showing the promise of the LTC paradigm in adapting LLM agents to new tasks and environments with minimal human effort.

References

Akata, E., Schulz, L., Coda-Forno, J., Oh, S. J., Bethge, M., and Schulz, E. Playing repeated games with large language models. *arXiv preprint arXiv:2305.16867*, 2023.

AntonOsika. gpt-engineer. https://github.com/AntonOsika/gpt-engineer, 2023. GitHub repository.

Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022a.

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022b.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2023.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, E., Wang, X., Dehghani, M., Brahma, S., et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., and Su, Y. Mind2web: Towards a generalist agent for the web, 2023.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*, 2022.

Hong, S., Zheng, X., Chen, J., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., and Wu, C. Metagpt: Meta programming for multi-agent collaborative framework, 2023.

Honovich, O., Scialom, T., Levy, O., and Schick, T. Unnatural instructions: Tuning language models with (almost) no human labor. *arXiv preprint arXiv:2212.09689*, 2022.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Huang, J., Gu, S. S., Hou, L., Wu, Y., Wang, X., Yu, H., and Han, J. Large language models can self-improve, 2022a.

Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., Sermanet, P., Brown, N., Jackson, T., Luu, L., Levine, S., Hausman, K., and Ichter, B. Inner monologue: Embodied reasoning through planning with language models. In *arXiv preprint arXiv:2207.05608*, 2022b.

Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S., and Grave, E. Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*, 2022.

Lee, H., Phatale, S., Mansoor, H., Lu, K., Mesnard, T., Bishop, C., Carbune, V., and Rastogi, A. Rlaif: Scaling reinforcement learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*, 2023.

Leike, J., Krueger, D., Everitt, T., Martic, M., Maini, V., and Legg, S. Scalable agent alignment via reward modeling: a research direction, 2018.

Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., and Ghanem, B. Camel: Communicative agents for ”mind” exploration of large scale language model society, 2023.

Lin, B. Y., Fu, Y., Yang, K., Ammanabrolu, P., Brahman, F., Huang, S., Bhagavatula, C., Choi, Y., and Ren, X. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *ArXiv preprint*, abs/2305.17390, 2023. URL https://arxiv.org/abs/2305.17390.

Liu, R., Yang, R., Jia, C., Zhang, G., Zhou, D., Dai, A. M., Yang, D., and Vosoughi, S. Training socially aligned language models in simulated human society. *arXiv preprint arXiv:2305.16960*, 2023a.

Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., Zhang, S., Deng, X., Zeng, A., Du, Z., Zhang, C., Shen, S., Zhang, T., Su, Y., Sun, H., Huang, M., Dong, Y., and Tang, J. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv: 2308.03688*, 2023b.

Longpre, S., Hou, L., Vu, T., Webson, A., Chung, H. W., Tay, Y., Zhou, D., Le, Q. V., Zoph, B., Wei, J., et al. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*, 2023.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Lu, P., Peng, B., Cheng, H., Galley, M., Chang, K.-W., Wu, Y. N., Zhu, S.-C., and Gao, J. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*, 2023.

Micheli, V. and Fleuret, F. Language models are few-shot butlers. *arXiv preprint arXiv:2104.07972*, 2021.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Muennighoff, N., Wang, T., Sutawika, L., Roberts, A., Biderman, S., Scao, T. L., Bari, M. S., Shen, S., Yong, Z.-X., Schoelkopf, H., et al. Crosslingual generalization through multitask finetuning. *arXiv preprint arXiv:2211.01786*, 2022.

OpenAI. Chatgpt. Online, 2022. URL https://openai.com/blog/chatgpt.

OpenAI. Gpt-4 technical report. *OpenAI blog*, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. Generative agents: Interactive simulacra of human behavior. In *In the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*, UIST '23, New York, NY, USA, 2023. Association for Computing Machinery.

Peng, B., Li, C., He, P., Galley, M., and Gao, J. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*, 2023.

Qian, C., Cong, X., Yang, C., Chen, W., Su, Y., Xu, J., Liu, Z., and Sun, M. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.

Qin, C., Zhang, A., Zhang, Z., Chen, J., Yasunaga, M., and Yang, D. Is chatgpt a general-purpose natural language processing task solver?, 2023.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. Direct preference optimization: Your language model is secretly a reward model, 2023.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21 (140):1–67, 2020.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Santacroce, M., Lu, Y., Yu, H., Li, Y., and Shen, Y. Efficient rlhf: Reducing the memory usage of ppo, 2023.

Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., Gallé, M., et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

Shi, W., Min, S., Yasunaga, M., Seo, M., James, R., Lewis, M., Zettlemoyer, L., and Yih, W.-t. Replug: Retrieval-augmented black-box language models. *arXiv preprint arXiv:2301.12652*, 2023.

Shinn, N., Cassano, F., Labash, B., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning, 2023.

Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., Zettlemoyer, L., and Fox, D. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10740–10749, 2020a.

Shridhar, M., Yuan, X., Côté, M.-A., Bisk, Y., Trischler, A., and Hausknecht, M. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020b.

Significant-Gravitas. Autogpt. https://github.com/Significant-Gravitas/AutoGPT, 2023. GitHub repository.

Stiennon, N., Ouyang, L., Wu, J., Ziegler, D., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. F. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33: 3008–3021, 2020.

Sumers, T., Yao, S., Narasimhan, K., and Griffiths, T. L. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*, 2023.

Talebirad, Y. and Nadiri, A. Multi-agent collaboration: Harnessing the power of intelligent llm agents. *arXiv preprint arXiv:2306.03314*, 2023.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm.stanford. edu/2023/03/13/alpaca. html*, 3(6):7, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Wang, R., Jansen, P., Côté, M.-A., and Ammanabrolu, P. Scienceworld: Is your agent smarter than a 5th grader?, 2022a. URL https://arxiv.org/abs/2203.07540.

Wang, X., Li, S., and Ji, H. Code4struct: Code generation for few-shot structured prediction from natural language. *arXiv preprint arXiv:2210.12810*, 2022b.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models, 2022c. URL https://arxiv.org/abs/2203.11171.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., and Zhou, D. Rationale-augmented ensembles in language models. *arXiv preprint arXiv:2207.00747*, 2022d.

Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N. A., Khashabi, D., and Hajishirzi, H. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2022e.

Wang, Y., Mishra, S., Alipoormolabashi, P., Kordi, Y., Mirzaei, A., Arunkumar, A., Ashok, A., Dhanasekaran, A. S., Naik, A., Stap, D., et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705*, 2022f.

Wang, Z., Mao, S., Wu, W., Ge, T., Wei, F., and Ji, H. Unleashing cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*, 2023.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., and Fedus, W. Emergent abilities of large language models, 2022a.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022b.

Weng, L. Llm-powered autonomous agents. *lilianweng.github.io*, Jun 2023. URL https://lilianweng.github.io/posts/2023-06-23-agent/.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., and Wang, C. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023a.

Wu, Y., Jiang, Z., Khan, A., Fu, Y., Ruis, L., Grefenstette, E., and Rocktäschel, T. Chatarena: Multi-agent language game environments for large language models. https://github.com/chatarena/chatarena, 2023b.

Xu, B., Peng, Z., Lei, B., Mukherjee, S., Liu, Y., and Xu, D. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

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

yoheinakajima. Babyagi. https://github.com/yoheinakajima/babyagi, 2023. GitHub repository.

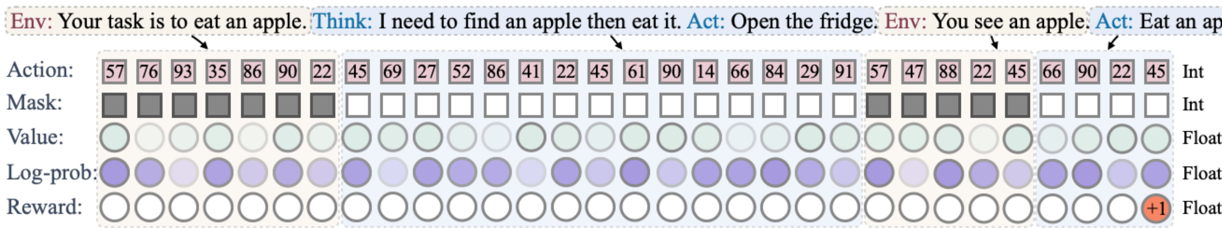

Figure 3. The buffer data is a serial of integer/float sequences. We treat each token id as the action in our reinforcement learning formula. We also save its corresponding mask indicating the source of the token, the value from the critic model, the log-prob indicating the log-likelihood when sampling the action and the reward from the environment/other agents.

A. Related Work

A.1. Instruction Tuning

Instruction tuning (IT) is an important technique for improving the capabilities and controllability of LLMs (Radford et al., 2019; Brown et al., 2020; Wei et al., 2022a; Qin et al., 2023; OpenAI, 2023; Chowdhery et al., 2022; Touvron et al., 2023). Many studies have been dedicated to instruction data generation and selection (Chung et al., 2022; Wang et al., 2022e; Lee et al., 2023). For instance, Unnatural Instructions (Honovich et al., 2022) is created by using the Super-Natural Instructions dataset (Wang et al., 2022f) as a seed to prompt InstructGPT (Ouyang et al., 2022). Self-Instruct (Wang et al., 2022e) employs a recursive pipeline that generates instruction data from hand-crafted seed tasks using ChatGPT (OpenAI, 2022). Other studies focus on fine-tuning pre-trained LLMs with instruction data. BLOOMZ (Muennighoff et al., 2022) is initialized with BLOOM (Scao et al., 2022) and then fine-tuned using the xP3 instruction dataset (Muennighoff et al., 2022). Flan-T5 is initialized with T5 (Raffel et al., 2020) and fine-tuned with the FLAN dataset (Longpre et al., 2023). Additionally, after the release of LLaMA (Touvron et al., 2023), many works have utilized it as the base model for instruction tuning, such as Alpaca (Taori et al., 2023), Vicuna (Chiang et al., 2023), and GPT-4-LLM (Peng et al., 2023). Some papers explore alignment fine-tuning using RLHF (Ouyang et al., 2022; Bai et al., 2022a; Stiennon et al., 2020; Leike et al., 2018). InstructGPT (Ouyang et al., 2022) employs GPT-3 for supervised fine-tuning on a human-filtered instruction dataset, followed by training a reward model and using PPO (Schulman et al., 2017) for RLHF. Claude investigates RLHF (Bai et al., 2022a) and constitutional approaches (Bai et al., 2022b) for making LLMs both harmless and helpful. DPO (Rafailov et al., 2023) fine-tunes the LLMs to align with human preferences by directly optimizing a classification problem on preference data instead of RLHF. While these prominent research works focus on aligning LLMs for general instruction-following, our objective is to adapt LLM agents for specific tasks or environments.

A.2. LLM Agents

LLMs have demonstrated the potential to act as advanced agents (Ouyang et al., 2022; Bubeck et al., 2023; Wei et al., 2022a), and significant progress has been made in developing versatile LLM agents (Weng, 2023; Sumers et al., 2023; Park et al., 2023; Liu et al., 2023a; Lin et al., 2023; Xu et al., 2023) and benchmarks (Wang et al., 2022a; Deng et al., 2023; Liu et al., 2023b). For planning, Chain-of-Thought (CoT(Wei et al., 2022b)) prompts the model to think step by step, by decomposing complex tasks into smaller and simpler steps. Self Consistency (Wang et al., 2022c;d) extends CoT by using ensembles of predictions to improve consistency of the LLM. Inner Monologue (Huang et al., 2022b) leverages environment feedback to enhance LLMs' planning and processing capabilities in embodied robotics tasks without extra training. ReAct (Yao et al., 2023) integrates reasoning and action taking, expanding the action space to include both task-specific discrete actions and language. Reflexion (Shinn et al., 2023) equips agents with dynamic memory and self-reflection capabilities to improve reasoning by using continuous trials in the same environment as feedback. Recent research has also shown that LLMs can be augmented as an autonomous agent to use *external tools* to solve problems in interactive environments. These techniques include retrieval augmentation (Shi et al., 2023; Yao et al., 2023; Izacard et al., 2022), math tools (Schick et al., 2023; Yao et al., 2023; Lu et al., 2023), and code interpreters (Gao et al., 2022; Wang et al., 2022b). Prior works also have explored using multiple LLMs in a collaborative setting to solve complex tasks (Hong et al., 2023; Qian et al., 2023; Li et al., 2023; Wang et al., 2023; Talebirad & Nadiri, 2023; Akata et al., 2023). Open-source projects like AutoGPT (Significant-Gravitas, 2023), GPT-Engineer (AntonOsika, 2023), and BabyAGI (yoheinakajima, 2023) also showcase the potential of LLM not just in generating content but also as a general problem solver. Most of the above methods are based on either human-designed

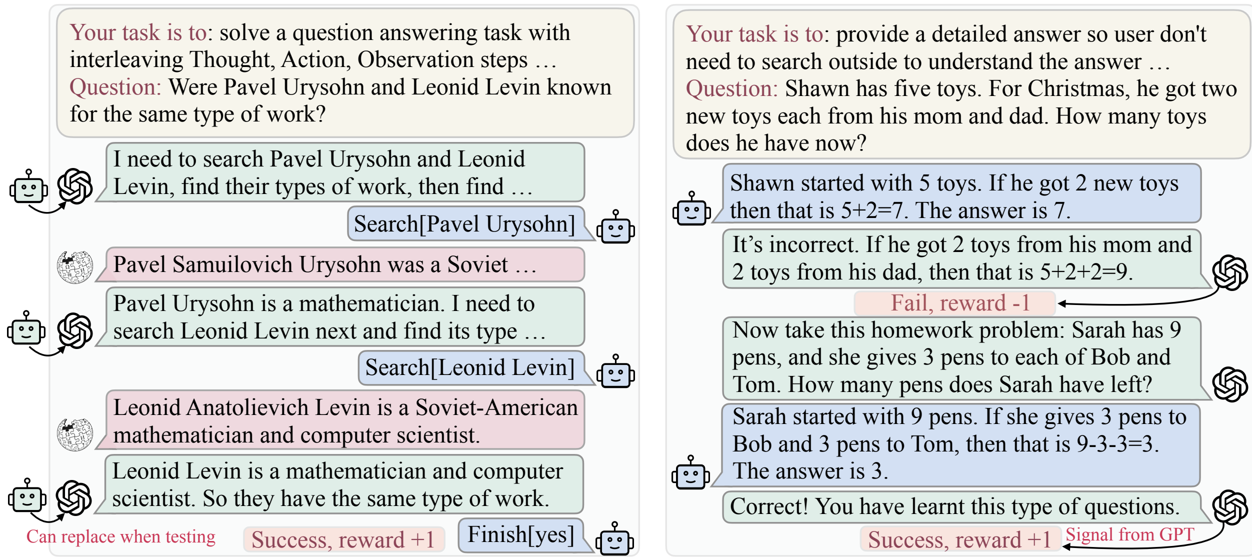

Figure 4. The toy examples to demonstrate communication patterns: 1) the left figure is the Multi-agent Dialogue pattern, where two agent play different roles to collaborate on the task. The thinker agent is responsible for analyzing the situation and give suggestion to the actor agent who is responsible for making decisions. We can just assign the LTC agent to play the thinker agent when testing without GPT-4 agent. 2) the right figure is the Teacher-student Dialogue pattern, where the student agent starts with an initial answer to the current question, and then the teacher directly corrects the answer with a reward. To help the student improve ability instead of just memorizing the solution, the teacher will generate another analogous question to ask the student. Eventually, the student gives a new answer for this analogous question and gets a new reward signal from the teacher.

few-shot prompting examples, or finetuning with pre-collected instruction datasets. Our LTC is not a few-shot prompting method and we focus on adapting the agent by collecting training data automatically by exploration.

B. Discussion

Method	GSM8k (CoT)	Hotpot-QA (ReAct)	Alfworld (ReAct)
ICL	836	1937	1744
LTC	**107**	**167**	**189**

Table 5. Average number of tokens of the input prompts on test sets. LTC does not use any few shot examples in the prompt, hence uses only a fraction of tokens compared to ICL.

Efficiency As mentioned above, prompting-based methods such as ReAct (Yao et al., 2023) and CoT (Wei et al., 2022b) use a subset of exemplary trajectories from the given task as few-shot prompts during inference. However, these few-shot prompts are often long, which leads to increased inference cost and limited context length for user queries. As shown in Table 5, we compare the number of input tokens for each task. We compute the CoT prompts for **GSM8k**, and we use ReAct for the other two tasks. All the few-shot prompts are sourced from the original paper. As shown, our LTC agents used only 12.8%, 8.6%, and 10.8% of the input tokens required by the ICL methods on the three tasks, respectively.

Shortcuts One interesting observation is that the GPT-4 agent sometimes employs ”shortcuts” to solve problems when serving as a teacher to generate new training data. These shortcuts rely on the internal knowledge acquired during its pretraining process. To illustrate this, we present a case study from *HotpotQA* in Figure 7. In this case, the GPT-4 agent quickly retrieves the answer by leveraging its memorized knowledge about the second entry after receiving the Wikipedia page of the first entry. On the other hand, the bottom of Figure 7 demonstrates a comparison with LLaMA-7B, which was trained using our LTC method with the GPT-4 agent in the loop. LLaMA-7B does not employ shortcuts and instead performs a search for the second entry. This case study demonstrates that communication mechanism in LTC provide additional benefits during learning, compared to soley relying on data generated by GPT-4.

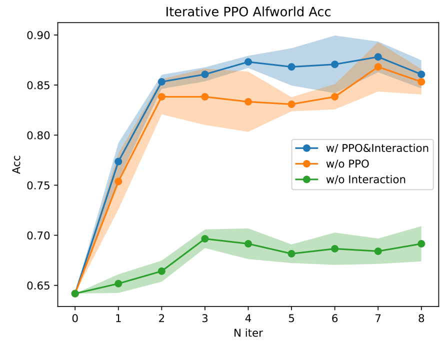

Figure 5. The accuracy curves of PPO training.

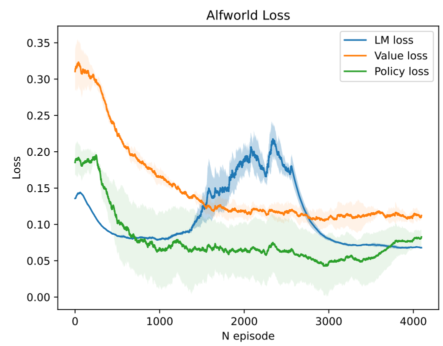

Figure 6. The loss curves of PPO training.

Ablation We conducted ablation studies on the loss design of LTC. Figure 5 illustrates the success rate of agents on the *ALFWorld* dataset under different loss settings. Without using our communication pattern for interactions and merely sampling pre-collected instruction data for training, the improvement was limited. However, when we incorporated our communication pattern to gather data, the model's performance quickly surpassed 80%. Furthermore, employing PPO loss to handle positive and negative samples separately resulted in faster and more significant improvement (blue line). In Figure 6, we present the separate curves of the three main losses during training. Initially, the LM loss showed a decreasing trend. Interestingly, as training iterations progressed, both the value loss and policy loss gradually decreased, which possibly causes the LM loss to increase temporarily. After the value loss and policy loss reached a certain threshold, the LM loss continued to decrease till convergence.

C. Appendix

C.1. Datasets

We conducted experiments on four datasets: *ALFWorld* (Shridhar et al., 2020b), *HotpotQA* (Yang et al., 2018), Chameleon (Wu et al., 2023b) and *GSM8k* (Cobbe et al., 2021). Each of these datasets represents a different environment type, namely single-agent, multi-agent collaborating, multi-agent competing, and teacher-student, respectively. And different communication patterns are used: Single-agent Monologue for *ALFWorld*, Multi-agent Dialogue for *HotpotQA* and Chameleon (Wu et al., 2023b), and Teacher-student Dialogue for *GSM8k*.

ALFWorld *ALFWorld* (Figure 1) is a text-based game that follows the ALFRED benchmark (Shridhar et al., 2020a). In this game, agents are presented with six types of tasks that involve navigating a simulated household environment using textual actions. With over 50 locations to explore, these tasks demand strategic planning and thorough exploration. Following (Shridhar et al., 2020b), we utilize the train set that consists of 3553 environments for training our model and the baselines; and we use the unseen test set that comprises 134 environments for evaluatation.

HotpotQA *HotpotQA* is a question-answering dataset that focuses on multi-hop reasoning based supporting facts, with the goal of improving the explainability of QA systems. In this dataset, agents are required to reason across two or more Wikipedia passages to derive answers. We initialize the environments using only the text of the questions, meaning that

Algorithm 1 The Python-style algorithm to demonstrate Monologue pattern

```
# agent: LLaMA agent
# input: Task description
# output: S = (T, M, R)

# initialization
T, M, R = [input], [0], [0]

i = 0
while i < max_steps:
   T += ["think:"]
   thought = agent.api(T)
   T.append(thought)
   M.append(1) # agent message mask
   R.append(0)


   T += ["act:"]
   action = agent.api(T)
   T.append(action)
   M.append(1) # agent message mask
   R.append(0)

   response = env.excute(action)
   reward = parse(response)
   T.append(response)
   M.append(0) # system message mask
   R.append(reward)

   i += 1
   if reward != 0:
      break
S = (T, M, R)
return S
```

agents are provided with the question and task description but do not have access to supporting paragraphs. To support their reasoning, agents must either rely on their internal knowledge or interact with an external Wikipedia tool to retrieve the necessary information. For training, we sample the environments from the training set, which consists of 90,447 QA-pairs. For evaluation, we run 500 random examples from the test set, following (Yao et al., 2023).

Chameleon Chameleon is a multi-player social deduction game environment implemented by the ChatArena (Wu et al., 2023b). There are two roles in the game, chameleon and non-chameleon. The topic of the secret word will be first revealed to all the players. Then the secret word will be revealed to non-chameleons. Non-chameleons try to identify the chameleon without giving away the secret word, while the chameleon tries to blend in and guess the word. The game involves giving clues, voting on who the chameleon might be, and a final guess from the accused chameleon. We use [3, 4, 5] players setting to train and test the agents' performance.

GSM8k The *GSM8k* dataset is a collection of 8.5K math problems for grade school students. These problems have been crafted by human experts to ensure linguistic diversity. The dataset is divided into two sets: 7.5K problems for training and 1K problems for testing. Each problem in the dataset requires 2 to 8 steps of reasoning to arrive at the solution. The problems primarily focus on fundamental arithmetic operations like addition, subtraction, multiplication, and division.

C.2. Communication Patterns

To collect the trajectories and the reward signal data from different types of tasks, we design the communication patterns for these tasks and unified the data format as described in Figure 3. Here we use three python-sytle algorithms (Algorithm 1 Algorithm 2 Algorithm 3) to demonstrate how three types of communication patterns help the agent collect exploration data.

C.3. Buffer Structure

The communication data will be saved as replay buffers for the updating phase, and the buffer data format is a serial of tokens sequences demonstrated in Figure 3. We treat each token as the action unit in our reinforcement learning formula, and each exploration trail is processed into 5 data sequences $[\mathbf{S_a}, \mathbf{S_m}, \mathbf{S_v}, \mathbf{S_l}, \mathbf{S_r}]$:

- $\mathbf{S_a}$: A list of integers representing the generated token ids encoded by the tokenizer. All the valid text trajectories are recorded as a queue, including system texts like environment descriptions, feedback, and agent texts like parsed actions,

Algorithm 2 The Python-style algorithm to demonstrate Dialogue pattern

```
# agent1: LLaMA agent
# agent2: GPT-4 agent
# input: Task description
# output: S = (T, M, R)

# initialization
T, M, R = [input], [0], [0]

i = 0
while i < max_steps:
   T += ["think:"]
   thought = agent2.api(T)
   T.append(thought)
   M.append(2) # teacher agent message mask
   R.append(0)


   T += ["act:"]
   action = agent1.api(T)
   T.append(action)
   M.append(1) # student agent message mask
   R.append(0)

   response = env.excute(action)
   reward = parse(response)
   T.append(response)
   M.append(0) # system message mask
   R.append(reward)

   i += 1
   if reward != 0:
      break
S = (T, M, R)
return S
```

thinking processes, and hints from other agents. While the invalid generated text of the agent will be skipped, such as nonsense string and action text can not be parsed. These tokens are treated equally as the input for the LLM, but they have different masks to apply different losses.

- $\mathbf{S_m}$: The system mask to mask different types of input tokens to control the training loss. We set 0 as the default mask for system texts like environment descriptions, system feedback, and system prompts, the actions encoded from these kinds of texts are not actions we want the agent to learn, so they will be masked out both policy loss and value loss in the PPO algorithm. We set 1 as the mask for agents-generated tokens like the keywords of decisions and the thinking process, which are the main supervising objects of our reinforcement learning pipeline, so they will be assigned full policy loss and value loss. We set 2 as the mask for hints or feedback from other agents, which are the actions we also want our own agent to learn but without instant state values since they are not generated by our agent. So the tokens with mask 2 will be mask out only the value loss and supervised by the policy loss.
- $\mathbf{S_v}$: The state values corresponding to the actions obtained by the value head our the agent model. The value head is an addition layer inserted to the original pre-trained LLM architecture, we implement it by inserting a linear layer after the second-to-last LlamaDecoderLayer as the auxiliary output module and the output values are processed by a $tanh()$ function to keep it range inside $(-1, 1)$.
- $\mathbf{S_r}$: The rewards corresponding to the actions. The rewards are very sparse, most of the actions are zero-reward, and only when the current task is finished or the token length of the current buffer has just overflowed it will be non-zero value: +1 for positive, -1 for negative.

C.4. Algorithm of LTC

The implementation of LTC can be summarized as Algorithm 4, we unveil the structural framework that embodies the Learning Through Communication (LTC) paradigm, meticulously crafted to facilitate autonomous and progressive learning through iterative cycles. At the outset, the pre-trained Large Language Model (LLM) agent undergoes a fine-tuning phase to align its initial configuration with the intended learning tasks. Following this, an exploration phase is initiated where a batch of training environments is sampled and subsequently engaged by the agent to generate trial data asynchronously, leveraging the power of parallel computing across multiple GPUs to enhance efficiency. This newly generated data is synchronized across all GPUs to foster a cohesive learning base, which is stored in a replay buffer for further utilization. As

Algorithm 3 The Python-style algorithm to demonstrate Analogue pattern

```
# agent1: LLaMA agent
# agent2: GPT-4 agent
# input: Question description
# output: S = (T, M, R)

# initialization
T, M, R = [input], [0], [0]

i = 0
while i < max_steps:
   T += ["answer␣the␣question␣step␣by␣step:"]
   answer1 = agent1.api(T)
   query = T + answer1 + ["the␣answer␣is␣correct,␣yes␣or␣no?␣also␣gives␣a␣better␣answer"]
   response = agent2.api(query)
   reward, answer2 = parse(response)
   T.append(answer1)
   T.append(answer2)
   M.append(1) # student agent message mask
   M.append(2) # teacher agent message mask
   R.append(reward)
   R.append(+1) # assume teacher is correct


   query = query + response + ["please␣generate␣a␣similar␣qa␣pair␣to␣teach␣the␣student:"]
   response = agent2.api(query)
   new_question, teacher_answer = parse(response)
   new_question += "answer␣the␣question␣step␣by␣step:"
   student_answer = agent1.api(new_question)
   reward = parse(student_answer, teacher_answer)
   T.append(new_question + student_answer)
   M.append(1) # student agent message mask
   R.append(reward)

   i += 1

S = (T, M, R)
return S
```

the agent iterates through this process, it continually samples from this replay buffer during the updating phase, employing the Proximal Policy Optimization (PPO) algorithm in a distributed data parallel (DDP) setting to refine its strategies and adapt dynamically. This code snippet, therefore, encapsulates the innovative asynchronous and distributed nature of the LTC paradigm, marking a significant stride in fostering intelligent, adaptive, and collaborative artificial intelligence agents.

C.5. Training Loss

After exploration in each iteration, we update the replay buffer by incorporating the newly collected trajectories and then sample the most recent trajectories to train the parameters π_θ of the LLM agent. We design our training objective to combine: 1) the standard language modeling loss $\mathcal{L}_{\text{LM}}$, 2) the policy loss $\mathcal{L}_{\text{policy}}$, 3) the value loss $\mathcal{L}_{\text{value}}$, and 4) the entropy loss $\mathcal{L}_{\text{entropy}}$. The overall training objective is formulated as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{LM}} + \beta(\mathcal{L}_{\text{policy}} + \lambda\mathcal{L}_{\text{value}} + \mathcal{L}_{\text{entropy}})$$

where β and λ are weighting hyperparameters.

The different losses in the above are described as follows:

- The LM loss $\mathcal{L}_{\text{LM}}$ is defined as the cross entropy between the agent and its generations which have a positive reward, akin to self-improving model schemes (Huang et al., 2022a; Rafailov et al., 2023). By training on these generations, the agent is further encouraged to give generations which yield positive rewards.

Algorithm 4 Python-style code of LTC

```
# agent: Pre-trained LLM agent
# n_gpu: total number of GPUs
# env_cls: the class of environments
# n_gen: the generation size for one iteration
# n_train: the train size for one iteration

# initialization
agent = instruction_finetune(agent)
replay_buffer = []
i = 0
while i < max_iteration:
  i += 1
  # Exploration Phase
  envs = env_cls(sample(data, n_gen//n_gpu))
  # asynchronously generate
  new_buffer = generate_trials(agent, envs)
  # dist.gather and dist.broadcast
  new_buffer = sync_all_gpus(new_buffer)
  replay_buffer.append(new_buffer)

  # Training Phase
  rollouts = sample(replay_buffer, n_train))
  # distributed training with ppo
  agent = ppo_ddp_train(agent, rollouts)
```

Figure 7. GPT-4 can use shortcuts to solve the problem, while the LLaMA-7B agent cannot mimic it.

- The policy loss $\mathcal{L}_{\text{policy}}$ is introduced to supervise the agent's actions. The policy loss $\mathcal{L}_{\text{policy}}$ is calculated using a masked version of the surrogate objective defined in (Schulman et al., 2017) with advantage estimates $\hat{A}$,

$$\mathcal{L}_{\text{policy}}(\theta) = -\mathbb{E}[m_{\text{policy}} * \min(r(\theta)\hat{A}, \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}],$$

where $r(\theta)$ is the output probability ratio $r(\theta) = \frac{\pi_\theta(a \mid s)}{\pi_{\text{old}}(a \mid s)}$ of the agent with its previous version π_{old}. We define binary mask m_{policy} to mask out the encoded system message in PPO loss (marked by $S_m = 0$ in buffers C.3). For example, let $\{x_1, y_1, x_2, y_2, \dots x_n, y_n\}$ be a token buffer consisting of system messages $x_n \in X$ and agents' messages (include the target trained agent and the other teacher agents) π_θ output $y_n \in Y$, then the binary mask $m_{\text{policy}} = \{0, 1, 0, 1, \dots 0, 1\}$.

- The value loss is defined in (Schulman et al., 2017) as the mean squared error between calculated value and estimated advantages masked by another binary mask m_{value} (marked by $S_m = 1$ in buffers C.3). For example, let $\{z_1, y_1, z_2, y_2, \ldots z_n, y_n\}$ be a token buffer consisting of all other messages (except the agent-generated messages) $z_n \in X$ and trained agent-generated messages π_θ output $y_n \in Y$, then the binary mask $m_{\text{policy}} = \{0, 1, 0, 1, \ldots 0, 1\}$.

- $\mathcal{L}_{\text{entropy}}$ is an entropy bonus to ensure sufficient exploration, as suggested in past work (Williams, 1992; Mnih et al., 2016). This entropy is computed as a small negative factor times the entropy of the policy distribution : $\mathcal{L}_{\text{entropy}} = 0.01 \times \sum_a \pi_\theta(a|s) \log \pi_\theta(a|s)$.

C.6. Implementation detail

C.7. Asynchronously Distributed Generating

The exploration data is generated in an asynchronous style, so that the agent can handle the environments with open-end exploration space. The training data are pre-processed into interactive environments which are capable for agents to observe the states, take actions, and get immediate feedback. According to the number of GPU threads, these environments are divided into corresponding portions and then distributed to each GPU. Subsequently, these GPUs begin to explore these environments asynchronously in parallel with the same agent trained by the latest data. Since the lengths of the generated contents are varied and the interactions inside the environments are generally open-ended, the time cost for the agent to explore each environment is also varied, some GPU threads may process the data faster than others. A barrier is set for all the GPU threads so that the early finished GPU threads can wait for the others until the total accumulated buffers generated by the environments achieve a preset number S_g, which is the quantity of the new training buffers we want to add to the replay buffers in one iteration. After all the GPU threads reach the barrier, we get enough buffers then gather the buffers from each GPU thread and merge them together, and broadcast the new buffers to each GPU thread to update their local replay buffers. The updated replay buffers will be used in the updating phase for training the agents of the next iteration.

C.8. Baselines

ReAct (Yao et al., 2023) uses a subset of training cases as prompts for different tasks, in the format of thought-action-observation sequences. For knowledge-intensive reasoning tasks like *HotpotQA*, ReAct designs an action space that includes search, lookup, and finish actions, enabling the agent to interact with Wikipedia to retrieve necessary information. On the other hand, **ReAct-IM** adopts Inner Monologue (IM) (Huang et al., 2022b) style prompting. Chain-of-thought prompting (**CoT**) (Wei et al., 2022b), enhances the reasoning capabilities of Language and Vision models (LLMs) by generating a sequence of intermediate reasoning steps. This can be considered as a reasoning-only baseline of ReAct, excluding actions and observations. Additionally, **CoT-SC** (Wang et al., 2022c;d) is a follow-up work of CoT, serving as a self-consistency baseline. It is worth noting that most of these methods employ greedy decoding, except for BUTLER (Micheli & Fleuret, 2021), which utilizes beam search. Most of these methods focus on few-shot prompting, and different pre-trained models are used. To ensure a fair comparison, we include the additional baselines named ReAct-Tuning and CoT-Tuning by fine-tuning the LLaMA-7B model using the collected trajectories as fine-tuning data mentioned in 3.1. In addition, GPT-4 are not used in the test time, and all the results reported are obtained by the trained agent itself.

C.9. Losses

We conducted ablation studies on the loss design of LTC. Figure Figure 5 illustrates the success rate of agents on the *ALFWorld* dataset under different loss settings. Without using our communication pattern for interactions and merely sampling pre-collected instruction data for training, the improvement was limited. However, when we incorporated our communication pattern to gather data, the model's performance quickly surpassed 80%. Furthermore, employing PPO loss to handle positive and negative samples separately resulted in faster and more significant improvement (blue line). In Figure

System: Interact with a household to solve a task. Here is the task:
System: You are in the middle of a room. Looking quickly around you, you see a a countertop 1, a fridge 1, cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a drawer 3, a drawer 2, a drawer 1, a garbagecan 1, a microwave 1, a shelf 3, a shelf 2, a shelf 1, a sinkbasin 1, a stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1.
System: Your task is to: put an egg in the bowl.

(LLaMA) Think: To solve the task, you need to find and take an egg, then find a bowl to put the egg in. An egg is more likely to appear in fridge (1).
(LLaMA) Act: go to fridge 1
System: The fridge 1 is closed.
(LLaMA) Act: open fridge 1
System: You open the fridge 1. The fridge 1 is open. In it, you see a bowl 1, a cup 2, a cup 1, a egg 1, a lettuce 1, a mug 2, a plate 2, a potato 2, a potato 1, and a tomato 1.
(LLaMA) Act: take egg 1 from fridge 1
System: You pick up the egg 1 from the fridge 1.
(LLaMA) Think: Now you take an apple (1). Next, you need to find the bowl and put the egg in. A bowl is more likely to appear on the countertop.
(LLaMA) Act: go to the countertop 1
System: On the countertop 1, you see a bowl 1, apple 2, a apple 1, a mug 3, a potato 3, and a spatula 1.
(LLaMA) Act: Put egg 1 in/on bowl 1
System: Success, reward +1

Figure 8. The text version of the toy example in Figure 1 for Monologue pattern of ALFWorld.

Figure 6, we present the separate curves of the three main losses during training. Initially, the LM loss showed a decreasing trend. Interestingly, as training iterations progressed, both the value loss and policy loss gradually decreased, which possibly causes the LM loss to increase temporarily. After the value loss and policy loss reached a certain threshold, the LM loss continued to decrease till convergence.

C.10. Cases study

One interesting observations is that the GPT-4 agent sometimes employs ”shortcuts” to solve problems when serving as a teacher to generate new training data. These shortcuts rely on the internal knowledge acquired during its pretraining process. To illustrate this, we present a case study from *HotpotQA* in Figure Figure 7. In this case, the GPT-4 agent quickly retrieves the answer by leveraging its memorized knowledge about the second entry after receiving the Wikipedia page of the first entry. On the other hand, the bottom of Figure Figure 7 demonstrates a comparison with LLaMA-7B, which was trained using our LTC method with the GPT-4 agent in the loop. LLaMA-7B does not employ shortcuts and instead performs a search for the second entry. This case study demonstrates that communication mechanism in LTC provide additional benefits during learning, compared to soley relying on data generated by GPT-4.