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]

[1]Georgia Institute of Technology, [2]Microsoft Azure AI, [3]Microsoft Research
Correspondence: kuanwang@gatech.edu, yeshe@microsoft.com

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 designing prompting methods (Wei et al., 2022b; Yao et al., 2023; Wu et al., 2023a), recent works also focus on how to train LLM agents using linguistic feedback and non-linguistic reward signals. The linguistic feedback is usually processed as instruction data to perform 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 align with human preferences (Ouyang et al., 2022; Bai et al., 2022a; Stiennon et al., 2020; Leike et al., 2018). Some existing approaches, such as Reflexion (Shinn et al., 2023) and ExpeL (Zhao et al., 2024), incorporate prior exploration experiences into the inference prompt to enhance decision-making. However, these methods introduce significant computational overhead during deployment due to increased prompt length. Reflexion, in particular, relies on ground truth feedback, which may lead to test set leakage concerns.

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 and Fleuret, 2021), while the reward signals serve solely as a filtering criterion to select the IFT data instead of as an objective of reinforcement learning. Moreover, since these methods primarily focus on inference-time modifications, they do not optimize the model's learning process in a structured way that balances linguistic consistency with task-specific rewards.

To address this gap, we propose *Learning through Communication* (LTC), a universal framework that enables LLM agents to simultaneously learn from linguistic feedback and non-linguistic reward signals (Figure 2). Our method organizes these signals into a unified structure and employs an iterative pipeline for continuous adaptation. Specifically, each iteration in LTC comprises two distinct phases:

1. **Exploration** During exploration, the agent interacts with various environments to gather diverse trajectories and reward signals into the universal buffer.

2. **Updating** The agent's model is updated based on the collected data in the universal buffer. LTC combines language modeling and reinforcement learning objectives to balance linguistic consistency with task-specific rewards.

Findings of the Association for Computational Linguistics: NAACL 2025, pages 6105–6122
April 29 - May 4, 2025 ©2025 Association for Computational Linguistics

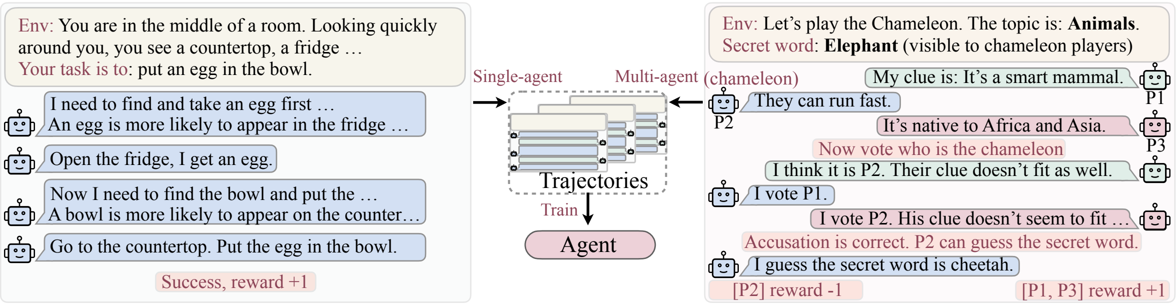

Env: You are in the middle of a room. Looking quickly around you, you see a countertop, a fridge …
Your task is to: put an egg in the bowl.
I need to find and take an egg first …
An egg is more likely to appear in the fridge …
Open the fridge, I get an egg.
Now I need to find the bowl and put the …
A bowl is more likely to appear on the counter…
Go to the countertop. Put the egg in the bowl.
Success, reward +1
Single-agent
Multi-agent
Trajectories
Train
Agent
Env: Let's play the Chameleon. The topic is: Animals.
Secret word: Elephant (visible to chameleon players)
(chameleon)
My clue is: It's a smart mammal.
P1
They can run fast.
P2
It's native to Africa and Asia.
P3
Now vote who is the chameleon
I think it is P2. Their clue doesn't fit as well.
I vote P1.
I vote P2. His clue doesn't seem to fit …
Accusation is correct. P2 can guess the secret word.
I guess the secret word is cheetah.
[P2] reward -1
[P1, P3] reward +1

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.

The buffer data format consists of a series of one-to-one aligned token-value sequences demonstrated in Figure 3. This replay buffer structure is designed to accommodate diverse tasks, including single-agent and multi-agent environments (Figure 1). To efficiently collect and organize agent interactions across varied scenarios, we employ a structured data format and an iterative policy update mechanism.

Buffer Structure The communication data is stored in replay buffers, facilitating systematic updates during the iterative policy learning process. Each exploration trajectory is processed into five aligned data sequences $[\mathbf{Sa}, \mathbf{Sm}, \mathbf{Sv}, \mathbf{Sl}, \mathbf{S_r}]$:

- **Sa**: Token sequence representing agent-generated actions, environment descriptions, system feedback, and hints from other agents. Invalid or nonsensical text is excluded.
- **Sm**: Mask for controlling training loss. Environment descriptions and system feedback (mask 0) are excluded from policy and value updates. Agent decisions and reasoning (mask 1) are fully supervised. Hints from other agents (mask 2) are included in policy loss but excluded from value updates.
- **Sv**: State values for actions, computed by the agent's value head. A linear layer in the LlamaDecoder processes these values, constrained within $(-1, 1)$ via a $tanh()$ function.
- **Sr**: Sparse rewards for actions. Most actions receive zero rewards, with nonzero values (+1 for positive, -1 for negative) assigned upon task completion or buffer overflow.

Iterative Policy Update Mechanism To facilitate online learning and adaptation, we introduce an iterative policy update pipeline that consists of two core phases:

1. **Exploration**: The agent interacts with the environment, collecting linguistic data and reward signals. These interactions are stored in the universal buffer, ensuring a structured representation of experiences for subsequent learning.
2. **Updating**: The agent updates its model based on the collected buffer data, leveraging both language modeling and reinforcement learning objectives. This phase integrates linguistic consistency with task-specific rewards, optimizing performance through policy and value learning.

Communication Patterns for Data Collection To effectively capture diverse agent interactions, we define three distinct communication patterns:

1. **Single-agent**: A single agent collects linguistic exploration data and receives environment-based rewards. The communication occurs between the agent and the environment. It follows a soliloquy-style interaction where the agent sequentially reasons and takes actions.
2. **Multi-agent**: Multiple agents collaborate or compete, exchanging linguistic data and utilizing environment-provided reward signals. Communication occurs among agents and the environment. Agents play distinct roles, either providing reasoning feedback or making decisions, and rewards are assigned based on collective or competitive performance.
3. **Teacher-student**: Experienced agents provide corrective feedback and reward signals

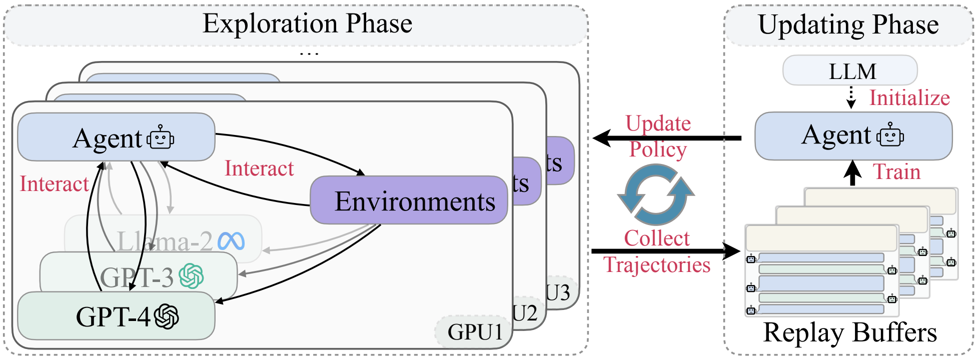


Figure 2: LTC has an iterative two-phase framework. During the exploration phase, the agent proactively explores new environments and communicates with other agents, gathering the trajectories to update the replay buffer. Then the agent is trained for updating the policy in the updating phase.

to novice agents. Unlike the previous patterns that rely primarily on environment-based rewards, this setting incorporates agent-provided feedback as additional reward signals. This approach is particularly effective for complex analytical tasks, where an experienced agent directly refines the learning process by generating structured rewards alongside guidance.

We evaluate the 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 a 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 the LTC approach across varied domains. The combination of a structured replay buffer, iterative policy updates, and diverse communication patterns enables Learning through Communication (LTC) to effectively balance linguistic feedback with task-driven reinforcement learning. Through evaluations on ALFWorld, HotpotQA, and GSM8k, our approach consistently outperforms supervised instruction tuning baselines, demonstrating its versatility and efficacy in facilitating LLM agent adaptation.

2 Experiments

2.1 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 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 and 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-

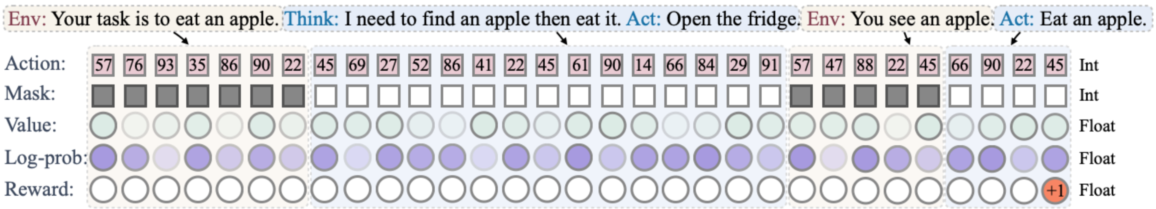


Figure 3: The buffer data is a serial of integer/float sequences. We treat each token id as the action in our reinforcement learning formula. We also save its corresponding mask indicating the source of the token, the value from the critic model, the log-prob indicating the log-likelihood when sampling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

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.

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 and Fleuret, 2021). The detailed of these baselines are described in Appendix D.8. 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. In addition, GPT-4 are not used in the test time, and all the results reported are obtained by the trained agent itself.

2.2 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 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.

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.

---

[*] 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).

[†] 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).

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)	31.0
	LTC(multi-agent)	**33.2**
Llama2-13B	ReAct-Tuning	33.8
	LTC(multi-agent)	**35.8**

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

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.

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.

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

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.

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)	39.6
	LTC(teacher-student)	**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.

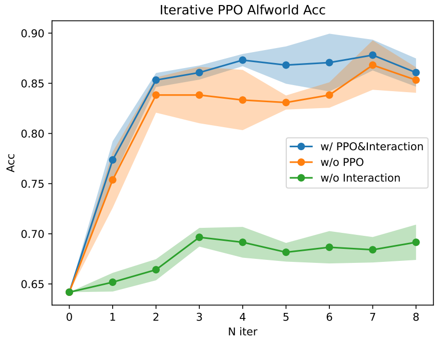


Figure 4: The accuracy curves of PPO training.

Ablation We conducted ablation studies on the loss design of LTC to show it balances linguistic feedback and reward signals. Figure 4 illustrates the success rate of agents on the *ALFWorld* 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%.

3 Conclusion

We introduced Learning through Communication (LTC), a paradigm that adapts LLM agents to new tasks and environments via communication-based iterative learning using both linguistic feedback and non-linguistic reward signals. Our evaluations across diverse domains like ALFWorld, HotpotQA, and GSM8k demonstrate LTC’s superiority over traditional methods, showcasing its ability to adaptively integrate comprehensive feedback for enhanced decision-making and reasoning. The designed communication patterns enhance LTC’s applicability across various settings, indicating its potential for broader, real-world applications and more natural human-agent interactions.

4 Limitation

Despite these advancements, our work has limitations. The LTC's dependence on diverse, high-quality feedback may not always be practical, limiting its robustness and broader applicability. Additionally, the need for careful hyper-parameter tuning, especially the balance between linguistic and non-linguistic feedback, can be challenging and time-consuming. Lastly, LTC's performance in highly dynamic or unpredictable environments remains underexplored, suggesting potential gaps in its ability to adapt to rapid changes or atypical feedback scenarios.

References

Elif Akata, Lion Schulz, Julian Coda-Forno, Seong Joon Oh, Matthias Bethge, and Eric Schulz. 2023. Playing repeated games with large language models. *arXiv preprint arXiv:2305.16867*.

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

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. 2022a. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*.

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Preprint*, arXiv:2306.06070.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *Preprint*, arXiv:2308.00352.

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2022a. Large language models can self-improve. *Preprint*, arXiv:2210.11610.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022b. Inner monologue: Embodied reasoning through planning with language models. In *arXiv preprint arXiv:2207.05608*.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*.

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

Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. 2018. Scalable agent alignment via reward modeling: a research direction. *Preprint*, arXiv:1811.07871.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for "mind" exploration of large scale language model society. *Preprint*, arXiv:2303.17760.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Prithviraj Ammanabrolu, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. 2023. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *ArXiv preprint*, abs/2305.17390.

Ruibo Liu, Ruixin Yang, Chenyan Jia, Ge Zhang, Denny Zhou, Andrew M Dai, Diyi Yang, and Soroush Vosoughi. 2023a. Training socially aligned language models in simulated human society. *arXiv preprint arXiv:2305.16960*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023b. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv: 2308.03688*.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*.

Vincent Micheli and François Fleuret. 2021. Language models are few-shot butlers. *arXiv preprint arXiv:2104.07972*.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, et al. 2022. Crosslingual generalization through multitask finetuning. *arXiv preprint arXiv:2211.01786*.

OpenAI. 2022. Chatgpt. Online.

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

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

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. 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. Association for Computing Machinery.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*.

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2023. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*.

Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. 2023. Is chatgpt a general-purpose natural language processing task solver? *Preprint*, arXiv:2302.06476.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Preprint*, arXiv:2305.18290.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. 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*, pages 627–635. JMLR Workshop and Conference Proceedings.

Michael Santacroce, Yadong Lu, Han Yu, Yuanzhi Li, and Yelong Shen. 2023. Efficient rlhf: Reducing the memory usage of ppo. *Preprint*, arXiv:2309.00754.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

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

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2023. Replug: Retrieval-augmented black-box language models. *arXiv preprint arXiv:2301.12652*.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020a. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020b. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

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

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

Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*.

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

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. 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.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022a. Scienceworld: Is your agent smarter than a 5th grader? *Preprint*, arXiv:2203.07540.

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

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022c. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2022d. Rationale-augmented ensembles in language models. *arXiv preprint arXiv:2207.00747*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022e. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. 2022f. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705*.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2023. Unleashing cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022a. Emergent abilities of large language models. *Preprint*, arXiv:2206.07682.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022b. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Lilian Weng. 2023. Llm-powered autonomous agents. *lilianweng.github.io*.

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

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023a. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Yuxiang Wu, Zhengyao Jiang, Akbir Khan, Yao Fu, Laura Ruis, Edward Grefenstette, and Tim Rocktäschel. 2023b. Chatarena: Multi-agent language game environments for large language models. `https://github.com/chatarena/chatarena`.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

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

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, pages 19632–19642. AAAI Press.

A 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.

A.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 $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$ represents the text data generated by the communication process during agent exploration, $\mathcal{M} = \{m_1, m_2, \dots, m_n\}$ with $m_i \in \{0, 1, 2\}$ indicates the source of the text data (system or agents), $\mathcal{R} = \{r_1, r_2, \dots, r_n\}$ 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 D.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**: Single-agent 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 Single-agent 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**: Multi-agent 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 5 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 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**: Teacher-student 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 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 5 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.

A.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:

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

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.

B Related Work

B.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.

B.2 LLM Agents

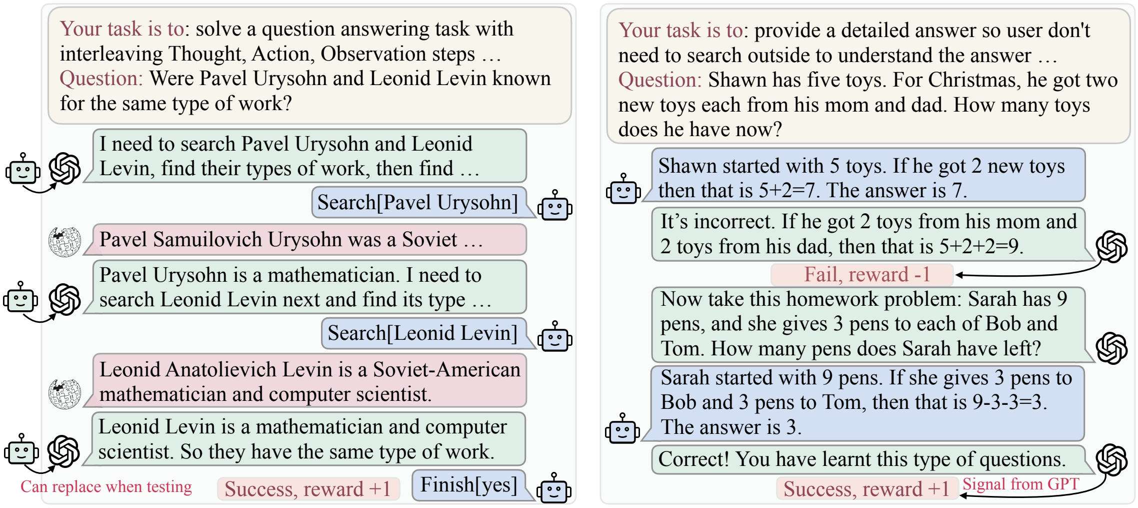
LLMs have demonstrated the potential to act as advanced agents (Ouyang et al., 2022; Bubeck et al., 2023; Wei et al., 2022a), and significant progress has been made in developing versatile LLM agents (Weng, 2023; Sumers et al., 2023; Park et al., 2023; Liu et al., 2023a; Lin et al., 2023; Xu et al., 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 and 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 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.

C 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



Your task is to: solve a question answering task with interleaving Thought, Action, Observation steps …
Question: Were Pavel Urysohn and Leonid Levin known for the same type of work?

I need to search Pavel Urysohn and Leonid Levin, find their types of work, then find …

Search[Pavel Urysohn]

Pavel Samuilovich Urysohn was a Soviet …

Pavel Urysohn is a mathematician. I need to search Leonid Levin next and find its type …

Search[Leonid Levin]

Leonid Anatolievich Levin is a Soviet-American mathematician and computer scientist.

Leonid Levin is a mathematician and computer scientist. So they have the same type of work.

Finish[yes]

Your task is to: provide a detailed answer so user don't need to search outside to understand the answer …
Question: Shawn has five toys. For Christmas, he got two new toys each from his mom and dad. How many toys does he have now?

Shawn started with 5 toys. If he got 2 new toys then that is 5+2=7. The answer is 7.

It's incorrect. If he got 2 toys from his mom and 2 toys from his dad, then that is 5+2+2=9.

Now take this homework problem: Sarah has 9 pens, and she gives 3 pens to each of Bob and Tom. How many pens does Sarah have left?

Sarah started with 9 pens. If she gives 3 pens to Bob and 3 pens to Tom, then that is 9-3-3=3. The answer is 3.

Correct! You have learnt this type of questions.

Figure 5: 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.

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.

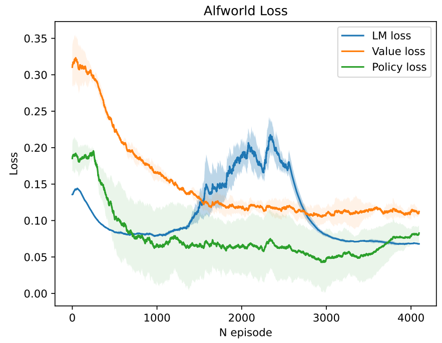


Figure 6: The loss curves of PPO training.

Ablation We conducted ablation studies on the loss design of LTC. Figure 4 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.

D Appendix

D.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 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.

Algorithm 1 The Python-style algorithm to demonstrate Single-agent 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
```

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.

D.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.

Algorithm 2 The Python-style algorithm to demonstrate Multi-agent 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
```

Algorithm 3 The Python-style algorithm to demonstrate Teacher-student 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
```

D.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, 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.

D.4 Algorithm of LTC

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)
```

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 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.

D.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:

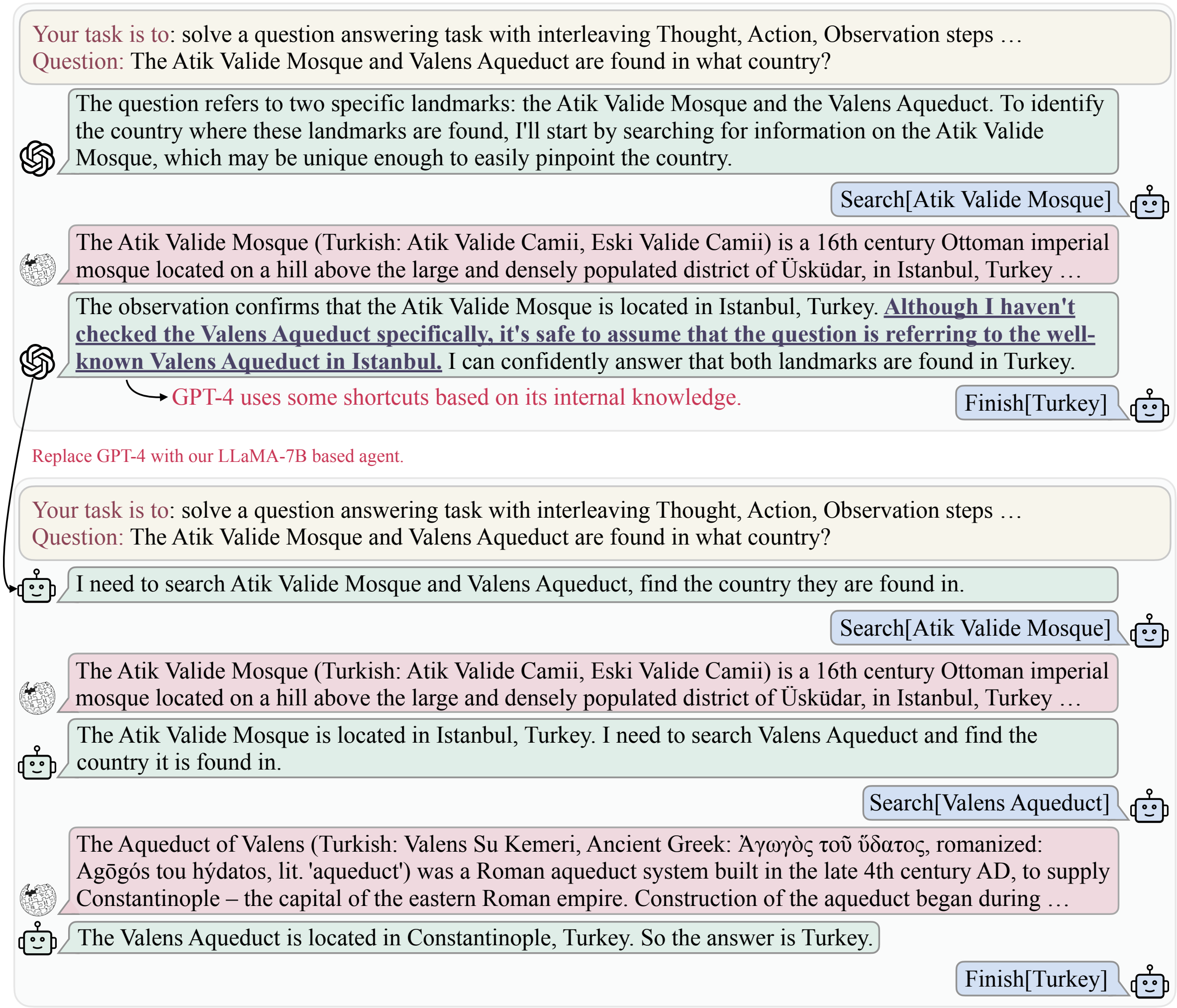
- The LM loss $\mathcal{L}_{\text{LM}}$ is defined as the cross entropy between the agent and its generations which have a positive reward, akin to self-improving model schemes (Huang et al., 2022a; Rafailov et al., 2023). By training on these generations, the agent is further encouraged to give generations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

- 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 $\hat{A}$,

  $$\begin{aligned} \mathcal{L}_{\text{policy}}(\theta) = &-\mathbb{E}[m_{\text{policy}} * \min(r(\theta)\hat{A}, \\ &\text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}], \end{aligned} \tag{2}$$

  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 D.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}



Your task is to: solve a question answering task with interleaving Thought, Action, Observation steps …
Question: The Atik Valide Mosque and Valens Aqueduct are found in what country?

The question refers to two specific landmarks: the Atik Valide Mosque and the Valens Aqueduct. To identify the country where these landmarks are found, I'll start by searching for information on the Atik Valide Mosque, which may be unique enough to easily pinpoint the country.

Search[Atik Valide Mosque]

The Atik Valide Mosque (Turkish: Atik Valide Camii, Eski Valide Camii) is a 16th century Ottoman imperial mosque located on a hill above the large and densely populated district of Üsküdar, in Istanbul, Turkey …

The observation confirms that the Atik Valide Mosque is located in Istanbul, Turkey. **Although I haven't checked the Valens Aqueduct specifically, it's safe to assume that the question is referring to the well-known Valens Aqueduct in Istanbul.** I can confidently answer that both landmarks are found in Turkey.

GPT-4 uses some shortcuts based on its internal knowledge.

Finish[Turkey]

Replace GPT-4 with our LLaMA-7B based agent.

Your task is to: solve a question answering task with interleaving Thought, Action, Observation steps …
Question: The Atik Valide Mosque and Valens Aqueduct are found in what country?

I need to search Atik Valide Mosque and Valens Aqueduct, find the country they are found in.

Search[Atik Valide Mosque]

The Atik Valide Mosque (Turkish: Atik Valide Camii, Eski Valide Camii) is a 16th century Ottoman imperial mosque located on a hill above the large and densely populated district of Üsküdar, in Istanbul, Turkey …

The Atik Valide Mosque is located in Istanbul, Turkey. I need to search Valens Aqueduct and find the country it is found in.

Search[Valens Aqueduct]

The Aqueduct of Valens (Turkish: Valens Su Kemeri, Ancient Greek: Ἀγωγὸς τοῦ ὕδατος, romanized: Agōgós tou hýdatos, lit. 'aqueduct') was a Roman aqueduct system built in the late 4th century AD, to supply Constantinople – the capital of the eastern Roman empire. Construction of the aqueduct began during …

The Valens Aqueduct is located in Constantinople, Turkey. So the answer is Turkey.

Finish[Turkey]

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

(marked by $S_m = 1$ in buffers D.3). For example, let $\{z_1, y_1, z_2, y_2, \dots z_n, y_n\}$ 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 $m_{\text{policy}} = \{0, 1, 0, 1, \dots 0, 1\}$.

- $\mathcal{L}_{\text{entropy}}$ 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 : $\mathcal{L}_{\text{entropy}} = 0.01 \times \sum_a \pi_\theta(a|s) \log \pi_\theta(a|s)$.

D.6 Implementation detail

D.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.

D.8 Baselines

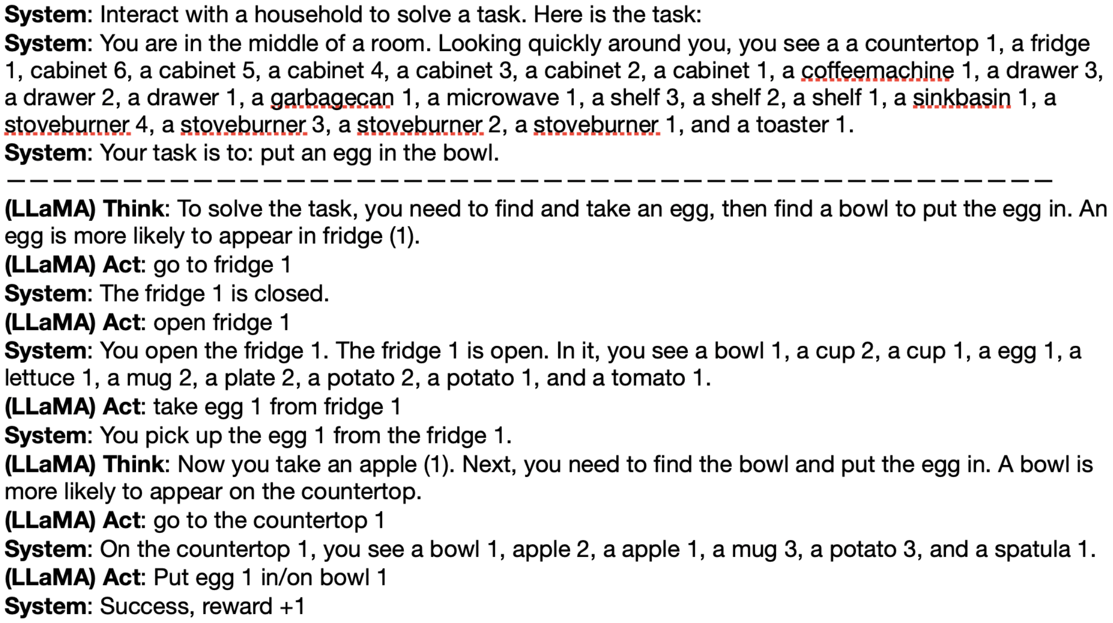
ReAct (Yao et al., 2023) uses a subset of training cases as prompts for different tasks, in the format of thought-action-observation sequences. For knowledge-intensive reasoning tasks like *HotpotQA*, ReAct designs an action space that includes search, lookup, and finish actions, enabling the agent to 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 and 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 2.1. In addition, GPT-4 are not used in the test time, and all the results reported are obtained by the trained agent itself.

D.9 Losses

We conducted ablation studies on the loss design of LTC. Figure Figure 4 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 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.

D.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.



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 Single-agent pattern of ALFWorld.