Task Execution via Reasoning, Tools and Conversations

By Daniel Huynh, Shiyi Liu, Zaiyi Zheng

ReAct: Synergizing Reasoning and Acting in Language Models

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, Yuan Cao

Synergizing Reasoning and Acting in Language Models

- Abilities of LLMs for reasoning and acting have been studied as separate topics
- Can we combine the ability to reason with taking actions as an agent?

ReAct

- **Re**asoning and **Act**ing
- Use LLMs to generate both verbal reasoning traces and task-specific actions together in interleaved manner
- Use reasoning traces to help model induce, track, update action plans and handle exceptions
- Actions allow it to interface with and gather additional info from external sources like knowledge bases

Modeling Human Intelligence

- Humans can seamlessly combine task-oriented actions with verbal reasoning as to why to take each action towards a goal

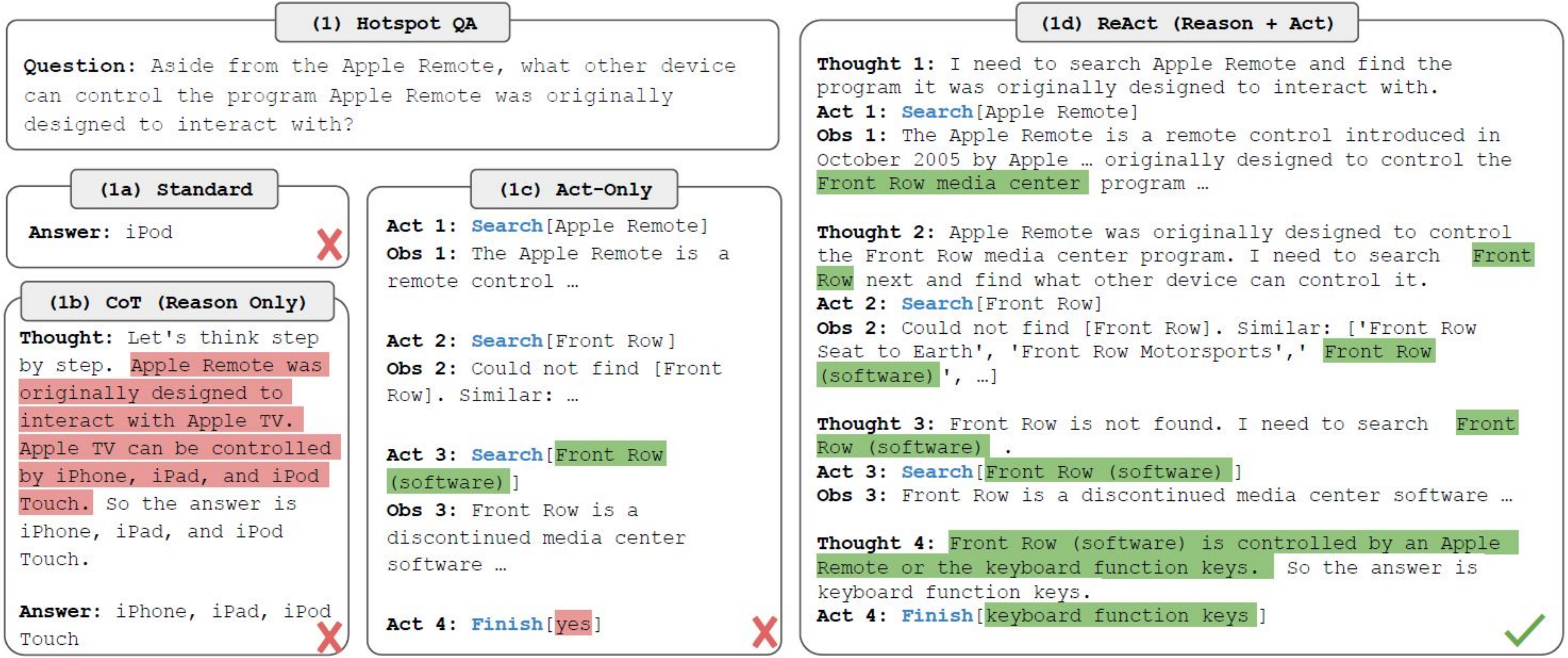
What this means:

- Allows one to learn new tasks quickly even under unseen circumstances or information uncertainties due to ability to reason
- Can deal with handling exceptions: adjust plans according to the situation or when things are missing in execution of a task
- Allows you to realize when external information is needed

Recent Works

- "Chain-of-thought" reasoning
  - Model uses own internal representations to generate thoughts, reasoning traces
  - However, no connection to the external world
    - Limits ability to reason reactively to diff situations and update knowledge
    - Very susceptible to fact hallucination and error propagation without grounding to external world
- Predicting Actions:
  - Do not use the LMs to reason abstractly about high-level goals, or maintain working memory to support acting

Example of ReAct in Action



Setup: Introducing Agents

- Agents:
  - Receive observation from environment, takes action that follows certain policy
  - Policy is based on choosing an action given context at time step t
  - Context contains all the previous observations and actions taken

- Agents of ReAct:
  - Augment agent’s action space to contain both Actions and the Language space
  - An action in the language space is referred to as a thought or reasoning trace
    - Does not affect external environment so there is no observation feedback
    - However, composes useful info by reasoning over current context and updates context to support future reasoning or acting

Setup: Introducing Agents (cont'd.)

- The *thought* may inject commonsense knowledge relevant to task solving or extract important parts from observations, track progress and transit action plans, and handle exceptions to adjust the action plans

Setup: Model

- Use frozen PaLM-540B
- Prompt with few-shot in-context examples to generate both domain-specific actions and free-form language tasks for task solving
- Each in-context example is a human trajectory of actions, thoughts, and environment observations to solve a task instance

Baselines

- Standard prompting (**STANDARD**): removes all thoughts, actions, observations in trajectories
- Chain-of-thought (**CoT**): removes actions and observations and serve as reasoning-only
- **CoT-SC**: self-consistency baseline that samples 21 CoT trajectories and adopts majority answer
- Acting-only prompt (**ACT**): removes thoughts, only acts

Evaluations

- Conduct empirical evaluations of ReAct and baselines on four diverse benchmarks:
  - HotPotQA (question answering)
  - Fever (fact verification)
  - ALFWorld (text-based game)
  - WebShop (webpage navigation)

Knowledge-Intensive Reasoning Tasks

- ReAct is able to retrieve info to support reasoning by interacting with a Wikipedia API, using reasoning to target what to retrieve next

Methods

- Randomly select 6 cases from training set and manually compose ReAct-format trajectories to use as few-shot exemplars for the prompts

Knowledge-Intensive Reasoning Tasks: Setup

- Two Datasets:
  - **HotPotQA** - multi-hop question answering benchmark requiring reasoning over two or more Wikipedia passages
  - **FEVER** - fact verification benchmark where claim is annotated with SUPPORTS, REFUTES, or NOT ENOUGH INFO based on whether exists Wikipedia passage to verify claim

- Action Space:
  - simple Wikipedia web API with three actions:
    - search[entity] → returns first 5 sentences from corresponding entity wiki page if exists or suggests top 5 similar entities
    - lookup[string] → return next sentence in the page containing string
    - finish[answer] → finishes the current task with *answer*

Combining Internal and External Knowledge

- Introduce two new baselines to test:
  - Incorporate both ReAct and CoT-SC, let model decide when to switch to the other method based on heuristics

- Choosing when to switch:
  - ReAct→CoT-SC when ReAct fails to return an answer within given number of steps
  - CoT-SC → ReAct when majority answer among n CoT-SC samples occurs less than n/2 times (internal knowledge might not support the task confidently)

Results

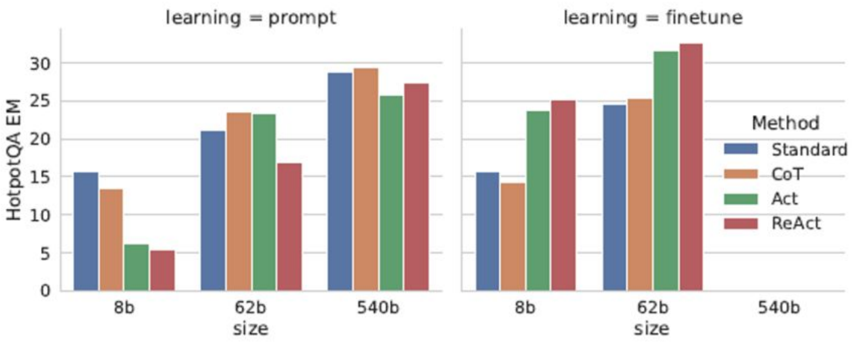
Prompt Method[a]	HotpotQA (EM)	Fever (Acc)
Standard	28.7	57.1
CoT (Wei et al., 2022)	29.4	56.3
CoT-SC (Wang et al., 2022a)	33.4	60.4
Act	25.7	58.9
ReAct	27.4	60.9
CoT-SC → ReAct	34.2	**64.6**
ReAct→ CoT-SC	**35.1**	62.0
Supervised SoTA[b]	67.5	89.5

Table 1: PaLM-540B prompting results on HotpotQA and Fever.

- ReAct outperforms ACT consistently
- ReAct outperforms CoT on Fever but slightly is behind CoT on HotPotQA
- ReAct + CoT-SC perform the best for prompting LLMs
  - Only takes 3-5 samples to reach CoT-SC performance with 21 samples
- ReAct performs best for fine-tuning
  - Scaling effect

Results: Scaling Effects

Scaling effect for prompting and fine-tuning on HotPotQA with ReAct:



- Fine-tuning helps ReAct and Act, and hurts Standard and CoT:
  - Standard and CoT simply learn to memorize potentially hallucinated knowledge facts when fine-tuned
  - ReAct and Act learn to reason and act to access info from Wikipedia when fine-tuned, which is a more generalizable skill for knowledge reasoning
- Prompting hurts ReAct due to difficulty to learn both reasoning and acting from in-context examples

Why did ReAct suffer on HotPotQA?

- Took random sample of trajectories and found:
  - Hallucination is problem for CoT, resulting in much higher false positive rate
  - Interleaving reason, action, and observation steps act as structural constraint reducing flexibility in reasoning
    - Error pattern where model repetitively generates previous thoughts and actions and goes in a loop, fails to reason the next action
  - When ReAct fails to find information it needs, it derails the model reasoning and has a hard time to recover
- Reveals tradeoff between factuality and flexibility

Decision Making Tasks

- Two Datasets:
  - **AlfWorld**
    - Text-based game, with 6 types of tasks where agent needs to achieve a high level goal by navigating and interacting with simulated household
    - Force agents to plan and track subgoals, learn likely locations for common household items
    - Test commonsense knowledge
  - **WebShop**
    - Proposed online shopping website environment with 1.18M real-world products and 12k human instructions
    - Requires agent to purchase product based on user instruction and act through web interactions
    - Ex: ex: “I am looking for a nightstand with drawers. It should have a nickel finish, and priced lower than $140”

AlfWorld

- **Prompting**
  - Each trajectory has thoughts that: 1) decompose goal, 2) track subgoal completion, 3) determine next subgoal, 4) reason via commonsense where to find an object and what to do with it
- **Baseline**
  - BUTLER
    - Imitation learning agent trained on 10^5 expert trajectories for each task type
  - Compare it with ACT and ReAct
- **Evaluation**
  - 134 unseen evaluation games in task-specific setup
  - Construct 6 prompts for each task type through permutation of 2 from the 3 we choose
  - ACT prompts chosen without the thoughts

WebShop

- **Evaluation**
  - Task evaluated by:
    - Average score (% desired attributes covered by chosen product averaged across all episodes)
    - Success rate (% episodes where chosen product satisfies all requirements)
    - On 500 test instructions

Results: AlfWorld

Method	Pick	Clean	Heat	Cool	Look	Pick 2	All
Act (best of 6)	88	42	74	67	72	**41**	45
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

Table 3: AlfWorld task-specific success rates (%). BUTLER and BUTLER_g results are from Table 4 of Shridhar et al. (2020b). All methods use greedy decoding, except that BUTLER uses beam search.

- Best ReAct trial has average success rate of 71%, beating best Act (45%) and BUTLER (37%) trials
- Worst ReAct trial (48%) beats best trial of both methods
- Act fails to correctly break goals into smaller subgoals, loses track of current state of environment without any thoughts

Results: WebShop

Method	Score	SR
Act	62.3	30.1
ReAct	**66.6**	**40.0**
IL	59.9	29.1
IL+RL	62.4	28.7
Human Expert	82.1	59.6

Table 4: Score and success rate (SR) on Webshop. IL/IL+RL taken from Yao et al. (2022).

- One-shot Act prompting already performs on par with IL and IL+RL methods
- Adding sparse reasoning to create ReAct achieves much better performance, with 10% improvement
- ReAct is more likely to identify instruction-relevant products/options by reasoning to bridge gap between observation noise and actions
- Still far from performance of expert humans - humans perform much more product exploration and query reformation

Investigating Internal Reasoning vs External Feedback

- Inner Monologue (IM) - previous work where actions from agent motivated by inner monologue
- IM is limited to observations of environment state, while ReAct’s reasoning traces is a lot more flexible and sparse, have diverse reasoning types
  - Internal reasoning vs simple reactions to external feedback
- Ablation Experiment: show differences between IM and ReAct reasoning
  - Create ReAct-IM, where prompts are in style of IM
  - ReAct substantially outperforms ReAct-IM:
    - 71% vs 53% success rate
  - ReAct-IM struggled to determine where items are due to lack of commonsense reasoning, as well as made mistakes in identifying when subgoals were finished and deciding the next subgoal

Benefits of ReAct

- Intuitive and easy to design
  - Human annotators just type down their thoughts in language on top of actions taken
- General and flexible
  - Due to flexibility of thought space and the thought-action occurrence format
- Performant and robust
- Human aligned and controllable
  - Easy to interpret decision making and reasoning process
  - Easy to inspect reasoning or factual correctness
  - Can correct or control agent behavior on the go by editing

Improvements

- Complex tasks with large action spaces require more demonstrations
- Learning from more high-quality human annotations will be big for furthering improving performance
- Scaling up ReAct with multi-task training or combining with RL could result in stronger agents

Toolformer: Language Models Can Teach Themselves to Use Tools

Timo Schick, Jane Dwivedi-Yu, Roberto Dess, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, Thomas Scialom

Observation

- LLMs exhibit remarkable abilities to solve new tasks from few shot examples.
- However, they paradoxically struggle with basic functionality (e.g. calculation).
- Ready-made tools can properly handle these basic functionalities.

Ready-made Tools

- Question Answering (based on another LM)
- Calculator
- Wikipedia Search
- Machine Translation System (based on another LM)
- Calendar

Challenges

- Lack of annotated corpus.
- Overfitting during the fine-tuning

Challenges

- Lack of annotated corpus.
- Overfitting during the fine-tuning

Intuition:

- LLMs should learn on their own how to correctly use tools while maintaining their generality.
    - On their own: No manual annotations
    - Correctly: *When* and *How*.
    - Generality: Maintain performance on other tasks.

Toolformer: How it works

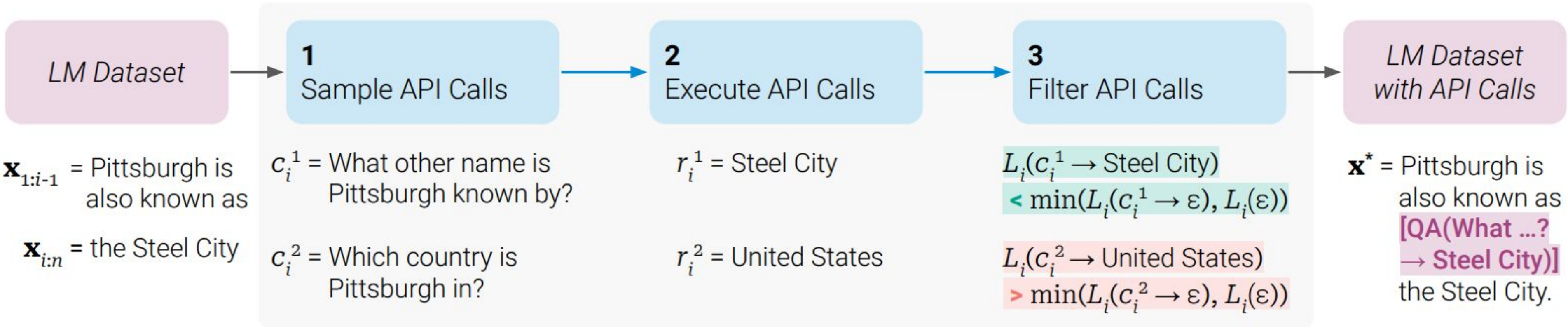
- **Motivation:** Generate *LM Dataset with API Calls*
- **API Call format:** $\mathrm{e}(c) = \texttt{<API>}\, a_c\, (i_c)\ \texttt{</API>}$

$$\mathrm{e}(c, r) = \boxed{\texttt{<API>}}\, \boxed{a_c}\, (\boxed{i_c})\ \boxed{\rightarrow}\, \boxed{r}\, \texttt{</API>}$$

<API> token API name API input ‘Request’ token return

Toolformer: How it works

- **Motivation:** Generate *LM Dataset with API Calls*
- **API Call format:** $$\mathrm{e}(c) = \texttt{<API>}\, a_c\,(i_c)\ \texttt{</API>}$$
$$\mathrm{e}(c, r) = \texttt{<API>}\, a_c\,(i_c)\ \rightarrow r\, \texttt{</API>}$$
- **Key steps:**



Toolformer: Sample API Calls

- **Motivation:** automatically insert API calls.
- **Basic idea:** ask GPT to generate potential API calls within the original text *x* using a few-shot prompt *P(x)*.

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of API calls:

Input: Joe Biden was born in Scranton, Pennsylvania.

Output: Joe Biden was born in **[QA("Where was Joe Biden born?")]** Scranton, **[QA("In which state is Scranton?")]** Pennsylvania.

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or **[QA("What other name is Coca-Cola known by?")]** Coke, is a carbonated soft drink manufactured by **[QA("Who manufactures Coca-Cola?")]** the Coca-Cola Company.

Input: x

Output:

Toolformer: Sample API Calls

- **Motivation:** automatically insert API calls.
- **Basic idea:** ask GPT to generate potential API calls within the original text *x* using a few-shot prompt *P(x)*.
- Select top-k positions by probabilities:

$$p_i = p_M(\texttt{<API>} \mid P(\mathbf{x}), x_{1:i-1})$$

We keep positions by a threshold:

$$I = \{i \mid p_i > \tau_s\}$$

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of API calls:

Input: Joe Biden was born in Scranton, Pennsylvania.

Output: Joe Biden was born in **[QA("Where was Joe Biden born?")]** Scranton, **[QA("In which state is Scranton?")]** Pennsylvania.

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or **[QA("What other name is Coca-Cola known by?")]** Coke, is a carbonated soft drink manufactured by **[QA("Who manufactures Coca-Cola?")]** the Coca-Cola Company.

Input: x

Output:

Toolformer: Filter API Calls

- **Motivation:** Solve the *When & How* problem of API calls' insertion.
- "Loss" function for API call **z**:

$$L_i(\mathbf{z}) = -\sum_{j=i}^{n} w_{j-i} \cdot \log p_M(x_j \mid \mathbf{z}, x_{1:j-1})$$

- Question: Why we put the API call **z** in front if the input text *x*?
- Answer: Make the task easier.

Toolformer: Filter API Calls

- **Motivation:** Solve the *When & How* problem of API calls' insertion.
- "Loss" function for API call **z**: $L_i(\mathbf{z}) = -\sum_{j=i}^{n} w_{j-i} \cdot \log p_M(x_j \mid \mathbf{z}, x_{1:j-1})$
-
- Positive "Loss" value after API call $\mathrm{e}(c_i, r_i)$: $L_i^+ = L_i(\mathrm{e}(c_i, r_i))$
- Negative "Loss" value without API call/return: $L_i^- = \min\left(L_i(\varepsilon), L_i(\mathrm{e}(c_i, \varepsilon))\right)$
- Filtering: Only keep API call at position i such that:

$$L_i^- - L_i^+ \geq \tau_f$$

Where τ_f is the threshold for filtering.

Model fine-tuning

- **Motivation:** Teach the model to use API calls while keeping generality.
- **New sequences:** x* = $x_{1:i-1}, \mathrm{e}(c_i, r_i), x_{i:n}$
- The original corpus *C* is a subset of the fine-tuning corpus *C**.

Experiments: Basic Setup

- **Methods & Baselines**
  - **GPT-J**
  - **GPT-J + CC:** GPT-J fine-tuned on a subset *C* of CCNET.
  - **Toolformer:** GPT-J fine-tuned on *C* with API calls.
  - **Toolformer (disabled):** Manually set the probability of <API> to be 0.
- **Tasks**
  - **LAMA:** Complete missing facts within each statement.
  - **Math:** Mathematical reasoning.
  - **Question Answering**

Experiments: Results

Model	SQuAD	Google-RE	T-REx
GPT-J	17.8	4.9	31.9
GPT-J + CC	19.2	5.6	33.2
Toolformer (disabled)	22.1	6.3	34.9
Toolformer	**33.8**	**11.5**	**53.5**
OPT (66B)	21.6	2.9	30.1
GPT-3 (175B)	26.8	7.0	39.8

LAMA: missing facts

Model	ASDiv	SVAMP	MAWPS
GPT-J	7.5	5.2	9.9
GPT-J + CC	9.6	5.0	9.3
Toolformer (disabled)	14.8	6.3	15.0
Toolformer	**40.4**	**29.4**	**44.0**
OPT (66B)	6.0	4.9	7.9
GPT-3 (175B)	14.0	10.0	19.8

Math: mathematical reasoning

Model	WebQS	NQ	TriviaQA
GPT-J	18.5	12.8	43.9
GPT-J + CC	18.4	12.2	45.6
Toolformer (disabled)	18.9	12.6	46.7
Toolformer	**26.3**	**17.7**	**48.8**
OPT (66B)	18.6	11.4	45.7
GPT-3 (175B)	29.0	22.6	65.9

Question Answering

Model	Es	De	Hi	Vi	Zh	Ar
GPT-J	15.2	**16.5**	1.3	8.2	**18.2**	**8.2**
GPT-J + CC	15.7	14.9	0.5	8.3	13.7	4.6
Toolformer (disabled)	19.8	11.9	1.2	10.1	15.0	3.1
Toolformer	**20.6**	13.5	**1.4**	**10.6**	16.8	3.7
OPT (66B)	0.3	0.1	1.1	0.2	0.7	0.1
GPT-3 (175B)	3.4	1.1	0.1	1.7	17.7	0.1

Multilingual Question Answering

Conclusion

- This paper proposes **Toolformer**, a language model that learns in a **self-supervised** way how to use different tools.
- Toolformer considerably improves zero-shot performance of small models to outperform large models (GPT-3).

Limitations:

- Inability of using tools in a chain.
- Does not support interactively using tools.
- Toolformer is sensitive to the exact wording of their input.
- Toolformer does not take into account the tool-dependent computational cost.

AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation

Authors: Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, Chi Wang
Affiliation: Microsoft Research, Pennsylvania State University, University of Washington, Xidian University

Introduction

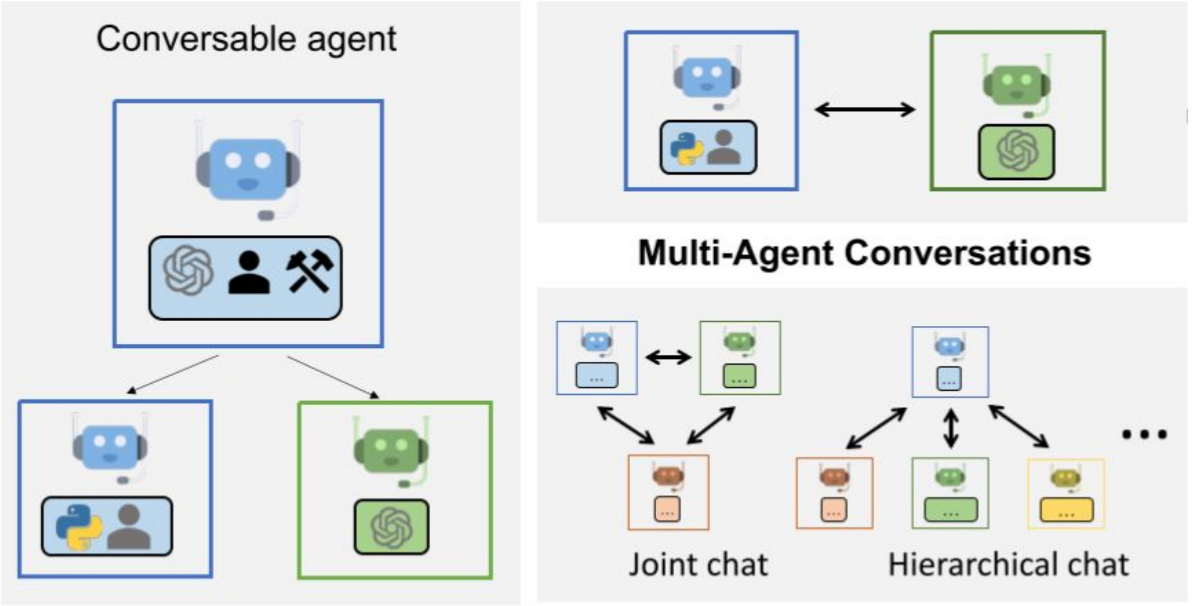
- Motivation: Expanding tasks that could benefit from LLMs and growing task complexity
- Intuition: Use multiple agents that cooperate through conversation
- Why multi-agent works:
    - Chat optimized LLMs (e.g., GPT-4) show the ability to incorporate feedback.
    - Single LLM can exhibit a broad range of capabilities, conversations between differently configured agents can help combine these broad LLM capabilities
    - LLMs have demonstrated ability to solve complex tasks when the tasks are broken into simpler subtasks
- Key Question: How to facilitate the development of LLM applications spanning broad domains and complexities based on a multi-agent approach?

AutoGen Framework

Two key concepts:

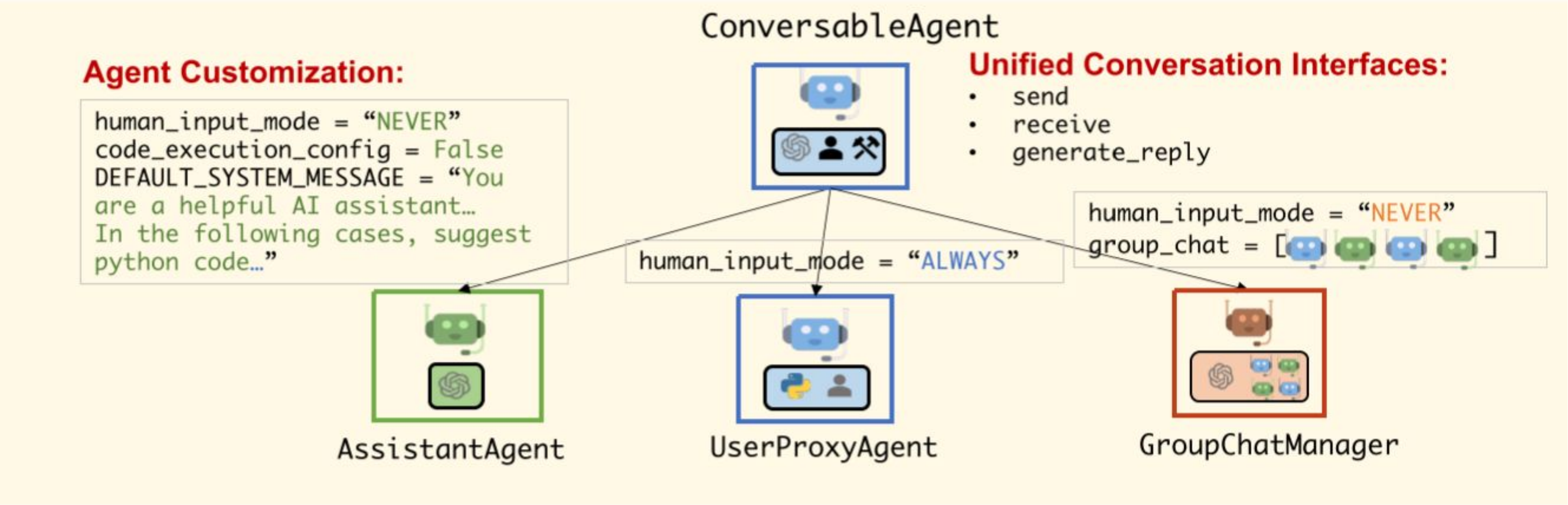
Conversable agents

Conversation programming



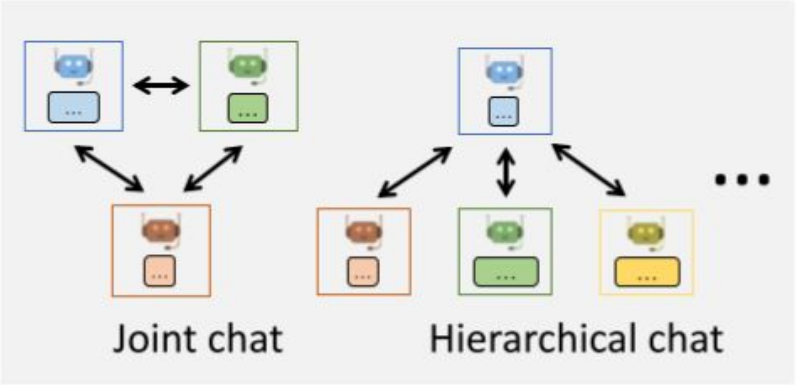
Conversable agents

- Generic design: Leverage LLMs, human inputs, tools, or a combination
- Customizable and reusable for different roles
- Conversable: Can receive, react, and respond to messages
- Enable human agency and automation



Conversation Programming

- Objective: Simplify and unify complex LLM application workflows as multi-agent conversations
- Two steps:
  - Defining conversable agents with specific capabilities and roles
  - Programming interaction behavior via conversation-centric computation and control
- Use both natural language and code to build applications with various conversation patterns



Flexible Conversation Patterns

Applications (Overview)

- AutoGen enables **diverse applications** demonstrating its effectiveness and flexibility
- Domains: Mathematics, coding, question answering, operations research, online decision-making, entertainment

Math problem solving

- Motivation: Leveraging LLMs for math problem-solving opens up new applications like **personalized AI tutoring and AI research assistance**
- AutoGen Approach:

  Scenario 1: Autonomous problem-solving using built-in agents

  Scenario 2: Human-in-the-loop problem-solving incorporating human feedback

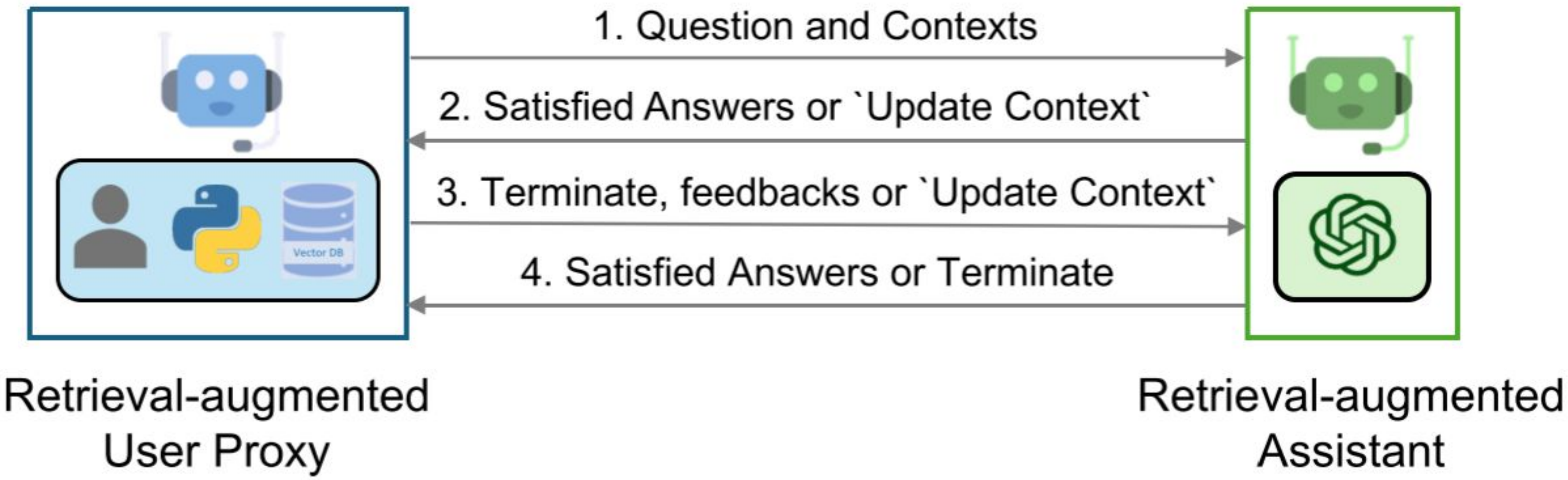
  Scenario 3: Multi-user problem-solving with a student and an expert collaborating with their assistant agents

- Results:

  On 120 level-5 problems from the MATH dataset, AutoGen achieves 52.5% accuracy, outperforming ChatGPT+Plugin (45.0%), ChatGPT+Code Interpreter (48.33%), GPT-4 (30.0%), Multi-Agent Debate (26.67%), and LangChain ReAct (23.33%)

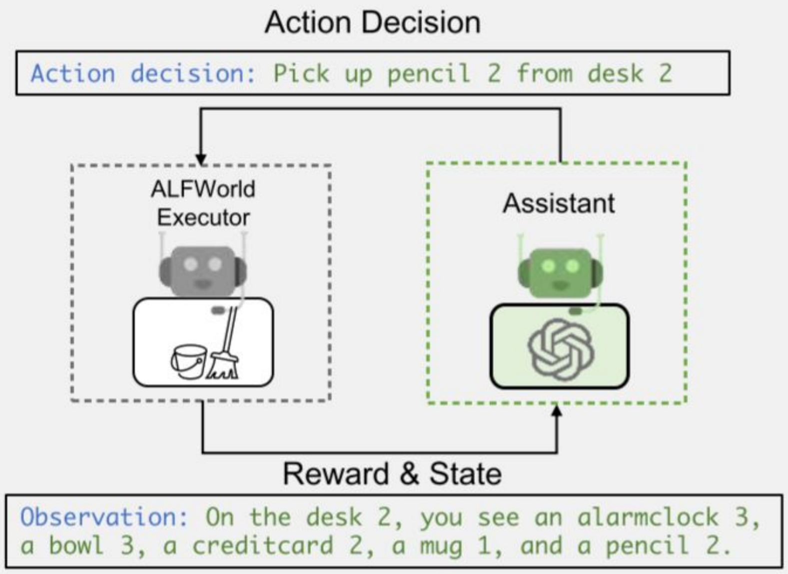
  On the entire MATH dataset, AutoGen achieves 69.48% accuracy, compared to GPT-4's 55.18%

Retrieval-Augmented Code Generation and Question Answering

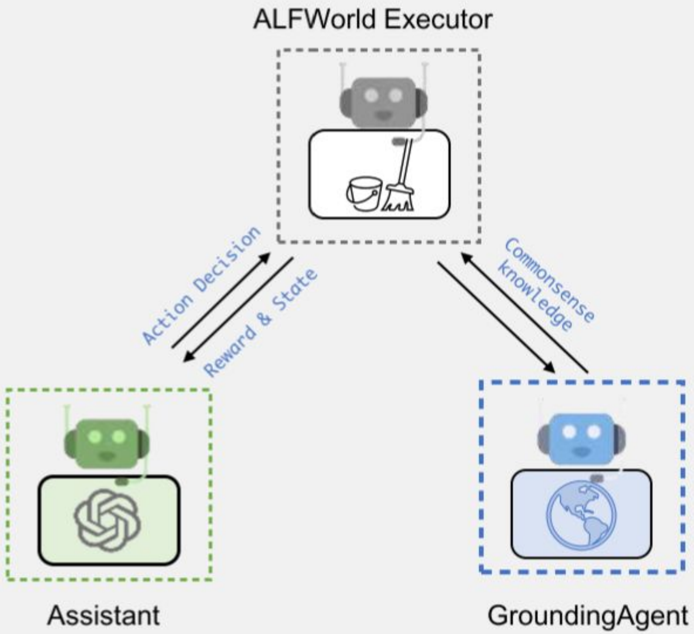


Scenario 2: Retrieval-augmented Chat successfully generates code using the latest codebase APIs not included in the LLM's training data

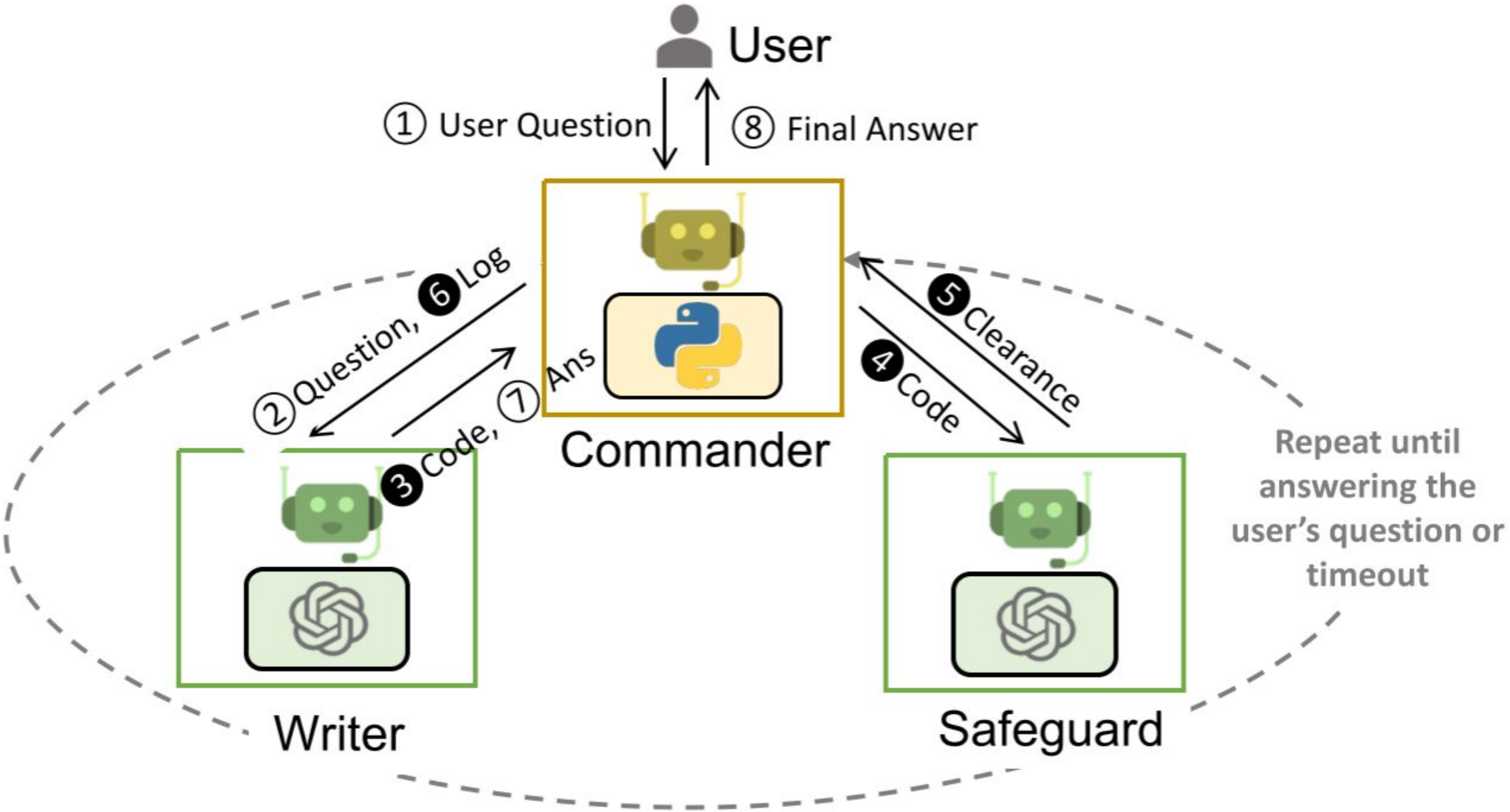
Decision Making in Text World Environments

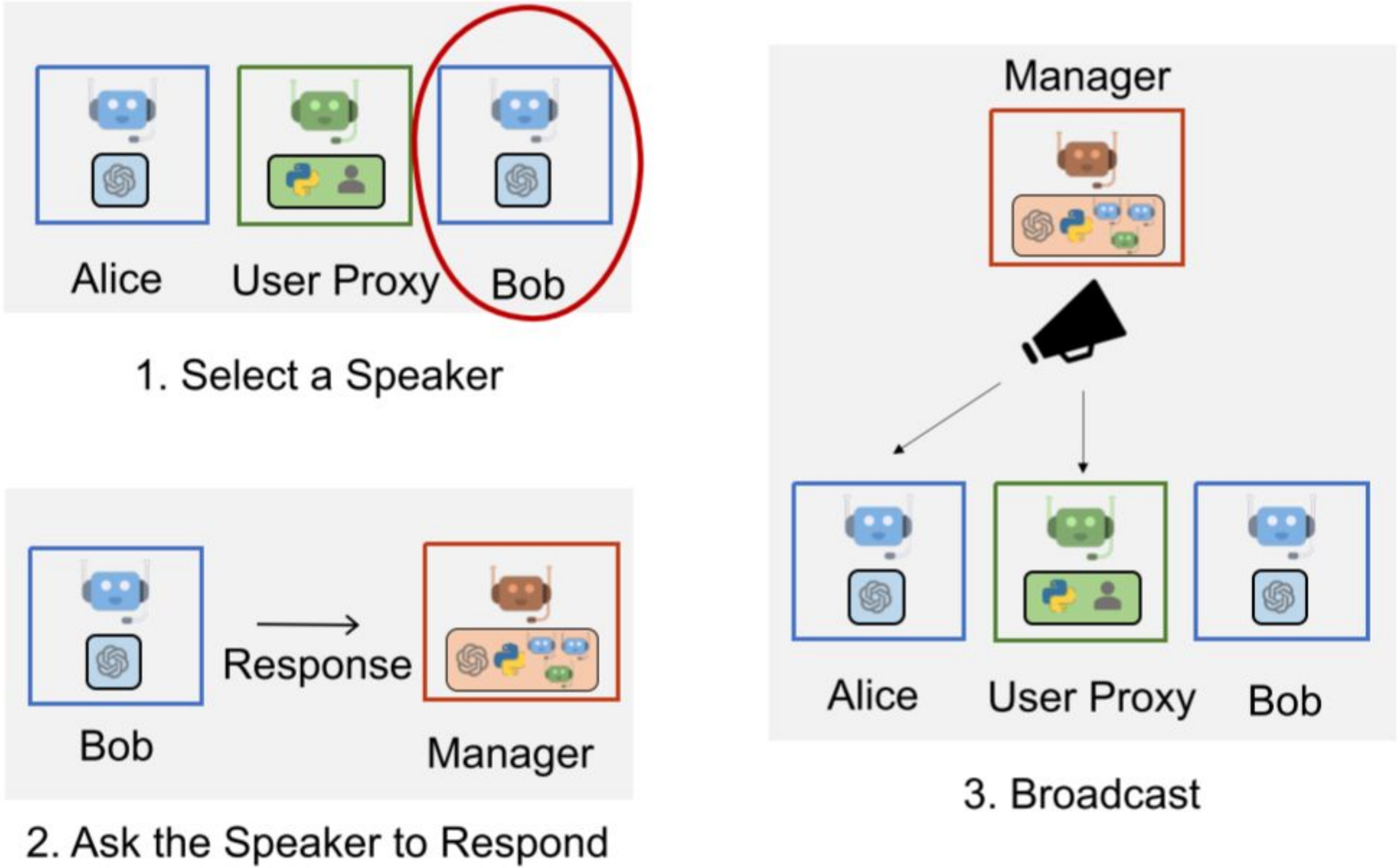


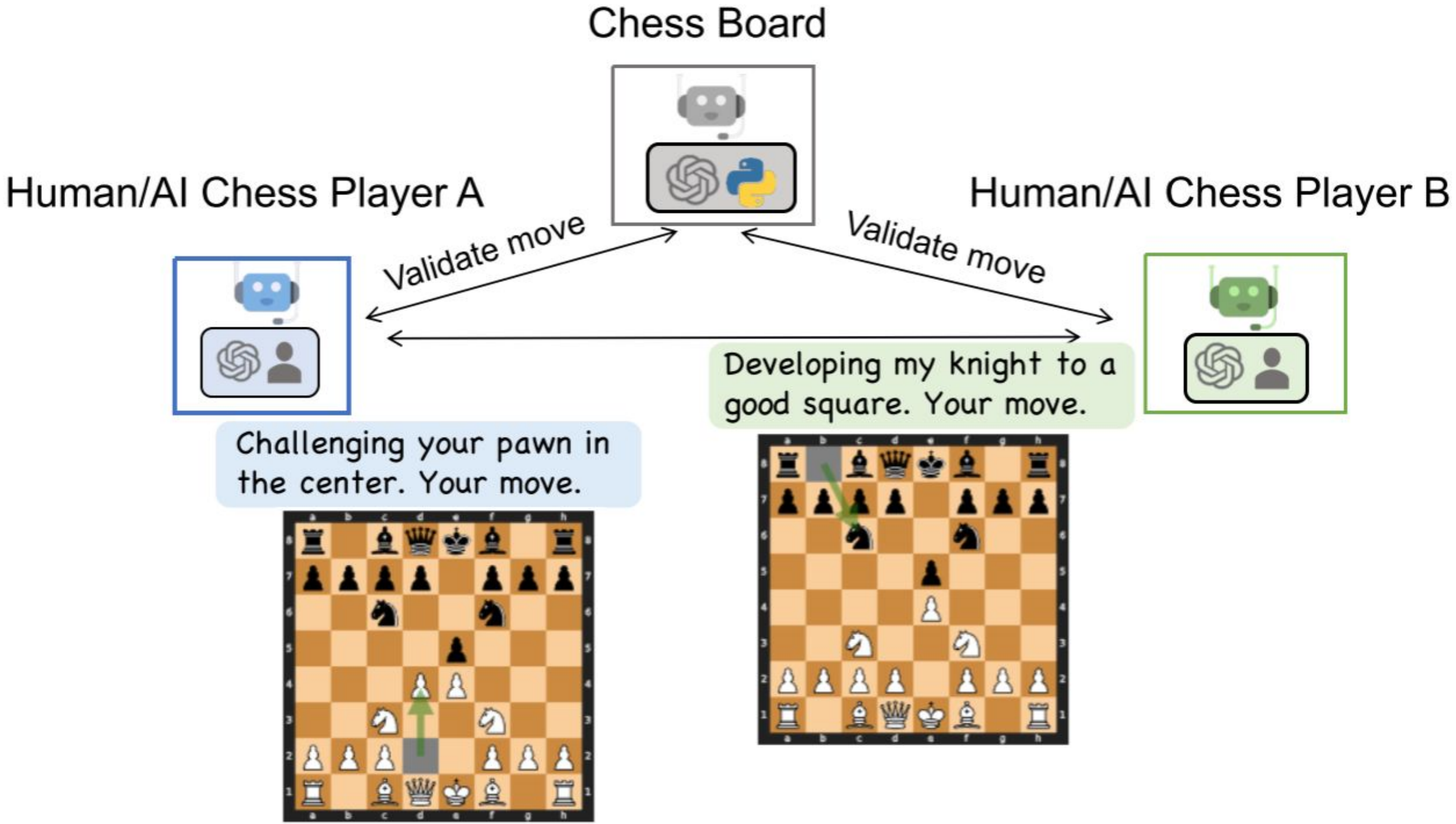
ALFChat (two agents)



ALFChat (three agents)


User
① User Question
⑧ Final Answer
Commander
② Question, 6 Log
3 Code, ⑦ Ans
4 Code
5 Clearance
Writer
Safeguard
Repeat until answering the user's question or timeout


Alice
User Proxy
Bob
1. Select a Speaker
Bob
Response
Manager
2. Ask the Speaker to Respond
Manager
Alice
User Proxy
Bob
3. Broadcast


Chess Board
Human/AI Chess Player A
Human/AI Chess Player B
Validate move
Validate move
Challenging your pawn in the center. Your move.
Developing my knight to a good square. Your move.

Benefits and Takeaways

- Improved performance, reduced development code, decreased manual burden
- Flexibility for developers, enabling dynamic conversation patterns
- Facilitates human participation alongside AI agents
- Modularity through separate agents

Future Directions

- Designing optimal multi-agent workflows for different tasks and applications
- Creating highly capable agents leveraging LLMs, tools, and humans
- Enabling scale, safety, and human agency in complex workflows

Reflexion: Language Agents with Verbal Reinforcement Learning

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, Shunyu Yao

Reflexion

- RL is expensive, require extensive training examples and model fine-tuning
- Instead with Reflexion, reinforce not through updating weights but through linguistic feedback – *verbal reinforcement*
- Convert binary or scalar feedback from environment into verbal feedback in form of textual summary
- This feedback acts as ‘semantic’ gradient signal

Advantages and Disadvantages vs RL

Advantages

- Lightweight and no fine-tuning
- More nuanced forms of feedback
- More explicit and interpretable form of memory over prior experiences (easy to understand and interpret steps)
- Provides explicit hints for future actions

Disadvantages

- Relies on LLM self-evaluation, so it does not have formal guarantee for success

Related Works

Related work on reasoning and decision-making					
Approach	Self refine	Hidden constraints	Decision making	Binary reward	Memory
Self-refine [15]	✓	✗	✗	✗	✗
Beam search [27]	✓	✓	✓	✓	✗
Reflexion (ours)	✓	✓	✓	✓	✓

- Previous work has employed self-evaluation, but some are limited to single-generation reasoning tasks
- Concepts can be enhanced with *self-reflection* to build persistent memory of self-reflective experiences
- Agent can identify its own errors and self-suggest lessons from mistakes

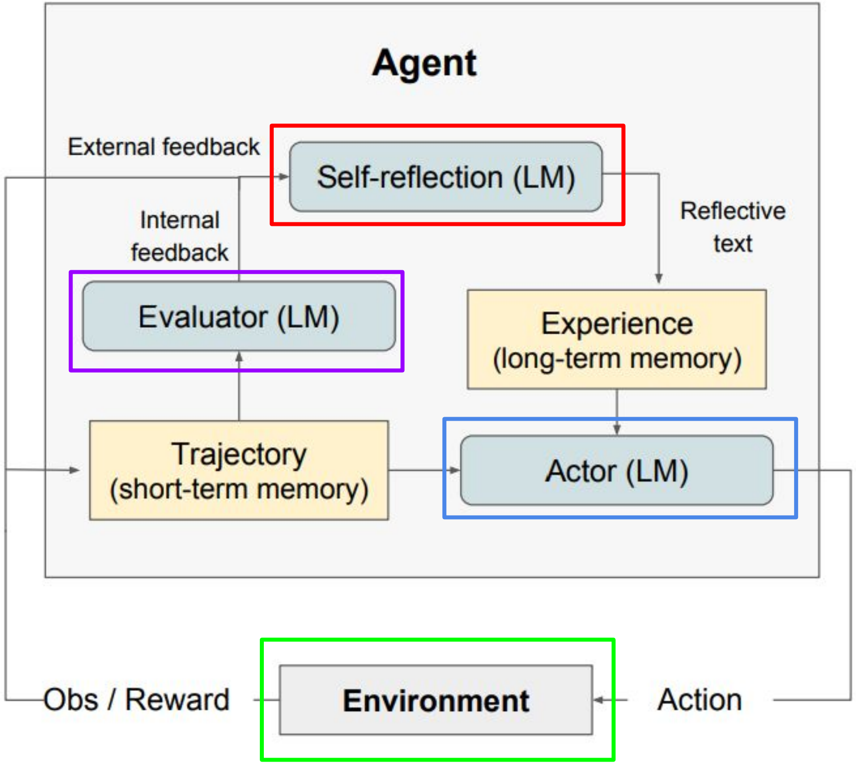
Related Works

Related work on programming

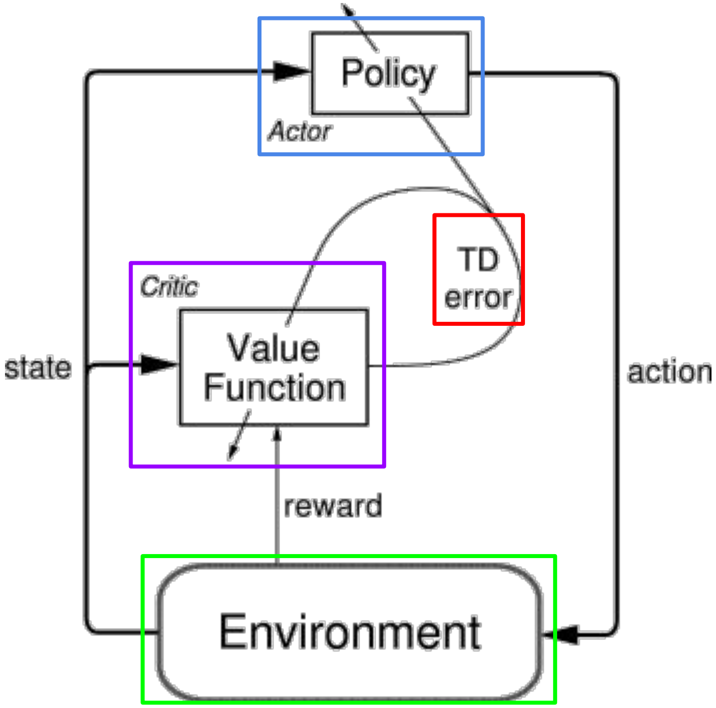
Approach Test execution	Test execution	Debugging	Self-generated tests	Multiple languages	Self-reflection
AlphaCode [14]	✓	✗	✗	✓	✗
CodeT [5]	✓	✗	✓	✗	✗
Self-debugging [7]	✓	✓	✗	✗	✗
CodeRL [12]	✓	✓	✗	✗	✗
Reflexion (ours)	✓	✓	✓	✓	✓

- None use *self-reflection* to bridge gap to learn from mistakes and improve
  - Simply can identify errors, but do not learn from them
- Most rely on ground truth test cases

Reflection: Overview by comparison with Actor-Critic



Reflection

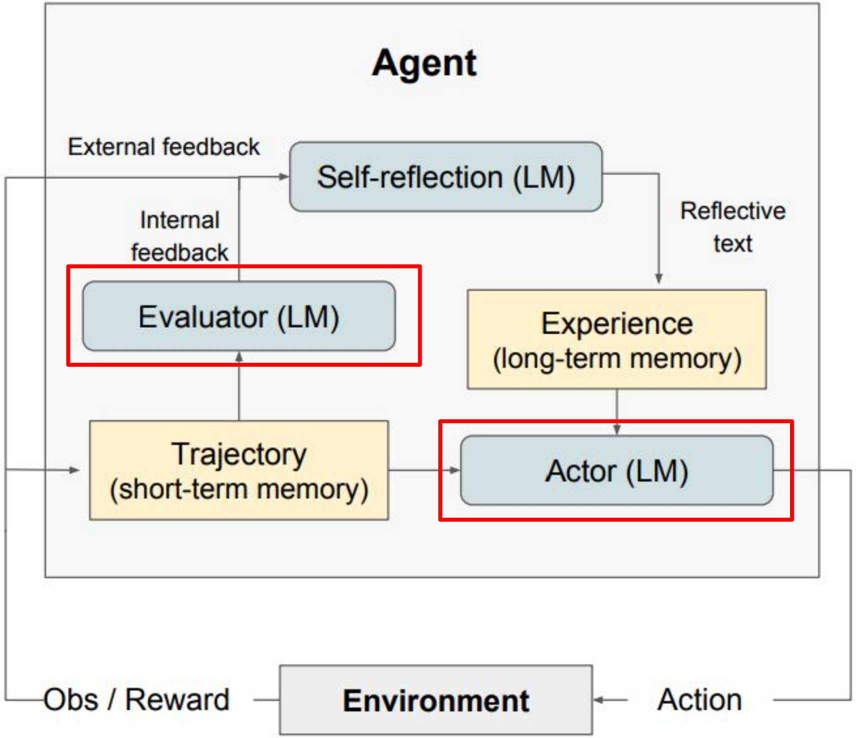


Actor-Critic Algorithm

Reflection: Actor & Evaluator

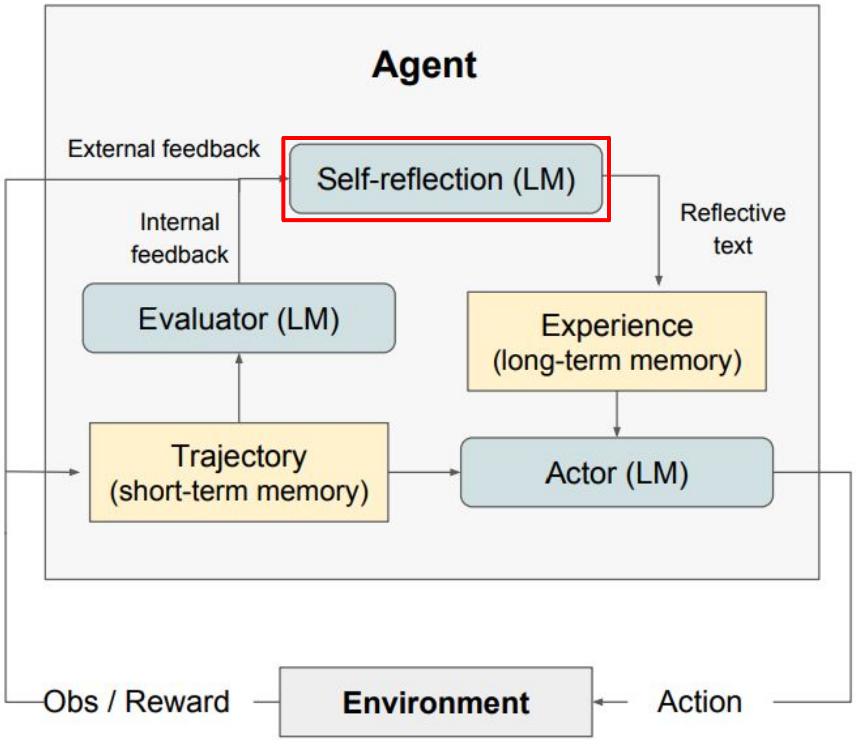
- **Actor:** generate text and actions
  - Model: ReAct, Chain-of-thought, etc.
  - Trajectory: multiple actions in a row.

- **Evaluator:** assessing the quality of Actor model's outputs.
  - Input: current trajectory
  - Reasoning tasks: exact match.
  - Decision-making: heuristic functions.
  - Different LLMs are tested as evaluators.



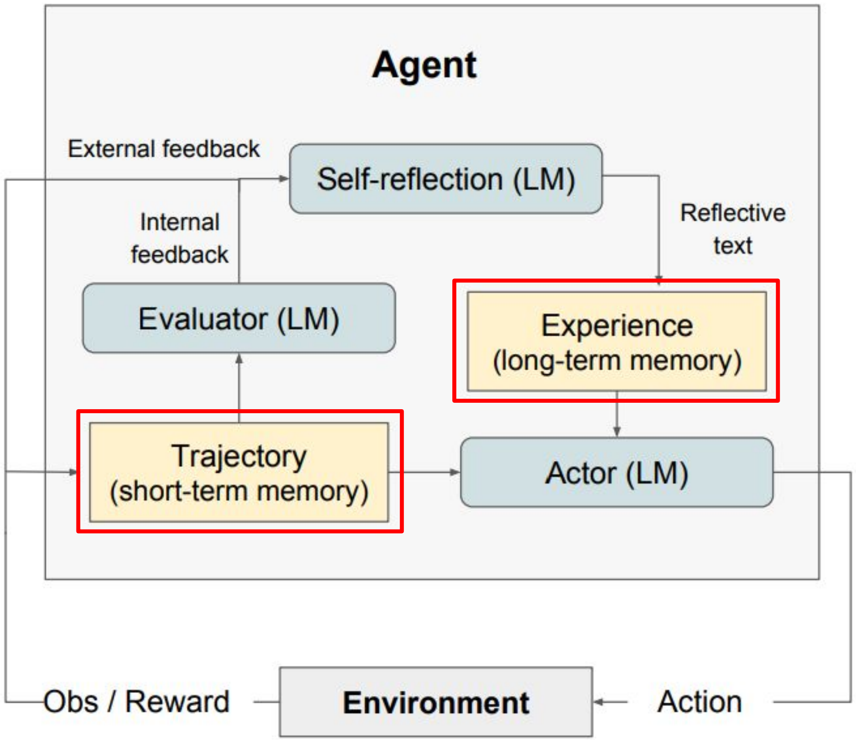
Reflection: Self-reflection

- Provides detailed feedback from binary Obs / Reward and trajectories.
- Feedback is stored in memory to inform future decisions.
- This process iteratively improves decision-making in diverse settings.



Reflection: Memory

- Reflexion integrates short-term and long-term memory.
- Short-term memory contains trajectory history.
- Long-term memory holds Self-Reflection model outputs.
- This dual-memory approach enhances contextual decision-making.



Experiments Overview

- Evaluate Reflexion on three types of tasks:
  - Decision-making tasks (ALFWorld) - test sequential action choices over long trajectories
  - Reasoning tasks (HotPotQA) - test knowledge-intensive, single-step generation improvement
  - Programming tasks (HumanEval, MBPP, LeetcodeHardGym) - teach the agent to effectively use external tools
- Compare Reflexion agents with strong baseline approaches

Decision-Making Results (ALFWorld)

- Dataset: ALFWorld - 134 tasks across six different household environments
- Metrics: Success rate (proportion of solved tasks)
- Measurement: Agent attempts each task for 12 iterative learning steps
- Results:
  - Reflexion achieves a success rate of 97% (130 out of 134 tasks) using a simple heuristic
  - Reflexion learns to solve additional tasks, improving from 85% to 97% success rate over 12 trials
  - Baseline (ReAct-only) performance increase halts between trials 6 and 7, reaching only 85% success rate

Reasoning Results (HotPotQA)

- Dataset: HotPotQA - 100 questions requiring multi-hop reasoning over Wikipedia articles
- Metrics: Exact match accuracy
- Measurement: Agent attempts each question until 3 consecutive failures or success
- Results:
  - Reflexion + ReAct: 51% accuracy, outperforming ReAct-only (39%)
  - Reflexion + Chain-of-Thought (CoT): 77% accuracy, outperforming CoT-only (57%)
  - Reflexion + CoT (given ground-truth context): 80% accuracy, outperforming CoT (given ground-truth context)-only (68%)
  - Ablation study: Self-reflection improves accuracy by 8% absolute over episodic memory alone (72% vs. 80%)

Programming Results (HumanEval, MBPP, LeetcodeHardGym)

- Datasets:
  - HumanEval: 164 Python programming problems
  - MBPP: 974 Python programming problems, 474 translated to Rust
  - LeetcodeHardGym: 40 challenging Leetcode problems in Python
- Metrics: Pass@1 accuracy (proportion of correct solutions on the first attempt)
- Measurement: Agent generates a solution and self-evaluates using test cases
- Results:
  - HumanEval (Python): Reflexion achieves 91.0% Pass@1 accuracy, outperforming the previous SOTA (80.1% by GPT-4)
  - HumanEval (Rust): Reflexion achieves 68.0% Pass@1 accuracy, outperforming GPT-4 baseline (60.0%)
  - MBPP (Python): Reflexion achieves 77.1% Pass@1 accuracy, underperforming GPT-4 baseline (80.1%) due to higher false positive rate (16.3% vs. 1.4% on HumanEval)
  - MBPP (Rust): Reflexion achieves 75.4% Pass@1 accuracy, outperforming GPT-4 baseline (70.9%)
  - LeetcodeHardGym (Python): Reflexion achieves 15.0% Pass@1 accuracy, outperforming GPT-4 baseline (7.5%)

Conclusion and Limitations

- Propose Reflexion: a novel framework for "verbal" reinforcement using linguistic feedback
- Significantly outperforms baselines across diverse tasks, demonstrating the effectiveness of self-reflection in LLMs
- Limitations:
  - May succumb to non-optimal local minima solutions
  - Memory component limited to a sliding window with maximum capacity
  - Test-driven development may face challenges in specifying accurate input-output mappings for certain functions

Future works

- Extend the **memory component** with advanced structures (e.g., vector embedding databases, SQL databases)
- Explore more **advanced RL techniques** in natural language (e.g., value learning, off-policy exploration)
- Address **safety and ethical considerations** when empowering autonomous agents
- Investigate how "verbal" reinforcement learning can make autonomous agents **more interpretable and diagnosable**

Common usages

- ReAct: Synergizing Reasoning and Acting in Language Models
  - Agents can reason and act based on natural language instructions
  - Applications: Question answering, task completion, and interactive problem-solving
- Toolformer: Language Models Can Teach Themselves to Use Tools
  - Agents learn to use external tools through self-supervision and interaction
  - Applications: Tool-based problem-solving, API usage, and code generation
- AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation
  - Multiple agents collaborate through conversation to solve complex tasks
  - Applications: Math problem-solving, code generation, question answering, decision-making, and creative tasks (e.g., chess)
- Reflexion: Language Agents with Verbal Reinforcement Learning
  - Agents learn from past experiences using self-reflection and verbal feedback
  - Applications: Sequential decision-making, reasoning, and programming tasks

Key Insights and Future Directions

- Synergizing reasoning and acting:
  - ReAct demonstrates that **integrating reasoning and acting in language models** leads to improved performance on tasks requiring both understanding and interaction
  - Future work should explore more **advanced reasoning** strategies and **expand the range of actions** agents can perform
- Self-supervised learning and tool use:
  - Toolformer shows that language models can **teach themselves to use external tools** through **self-supervision and interaction**
  - This opens up new possibilities for agents to acquire skills and knowledge without explicit human guidance
  - Future research should investigate how to **scale up tool learning** and enable agents to discover and **use new tools autonomously**
- Multi-agent collaboration and conversation:
  - AutoGen introduces a framework for building applications using **multiple conversing agents**, each with specific capabilities and roles
  - This approach enables tackling **more complex tasks** and facilitates human-AI collaboration
  - Future work should explore **optimal multi-agent workflows**, create highly capable agents, and address challenges in scaling, safety, and human agency
- Verbal reinforcement learning and self-reflection:
  - Reflexion proposes a **novel "verbal" reinforcement learning** paradigm, where agents learn from linguistic feedback and self-reflection
  - This approach improves interpretability and alignment compared to traditional RL methods
  - Future research should extend the memory component, investigate **advanced RL techniques in natural language**, and address safety and ethical considerations