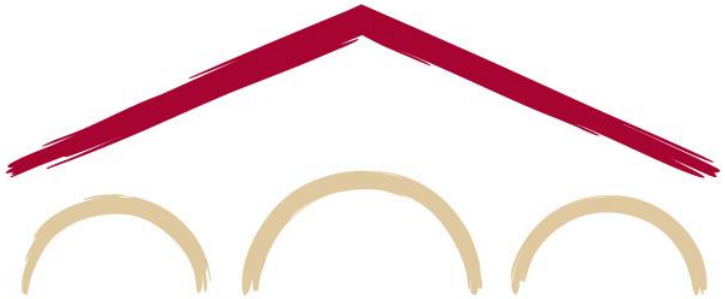
Natural Language Processing with Deep Learning
CS224N/Ling284



Diyi Yang

Lecture 10: RAG and Language Agents

Overview

From last lecture: PEFT--adapters (5 mins)

1. Question answering and RAG (15 mins)
2. Introducing language agents (5 mins)
3. Reasoning and planning (15 mins)
4. Memory (15 mins)
5. Tool use (10 mins)
6. Agent data and evaluation (15 mins)

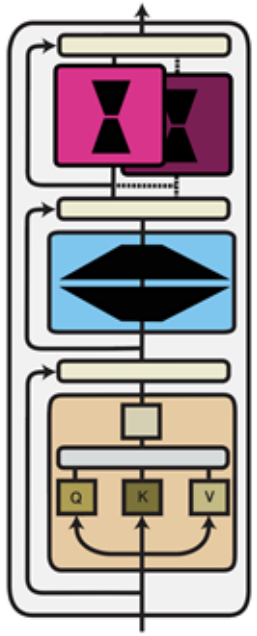
- Redeem credits! Select keywords for your project!
- Huggingface tutorial this Friday

A functional perspective of adaptation

- Function composition augments a model's functions with new task-specific functions:

$$f_i'(\boldsymbol{x}) = f_{\theta_i}(\boldsymbol{x}) \odot f_{\phi_i}(\boldsymbol{x})$$

- Most commonly used in multi-task learning where modules of different tasks are composed.



Function Composition

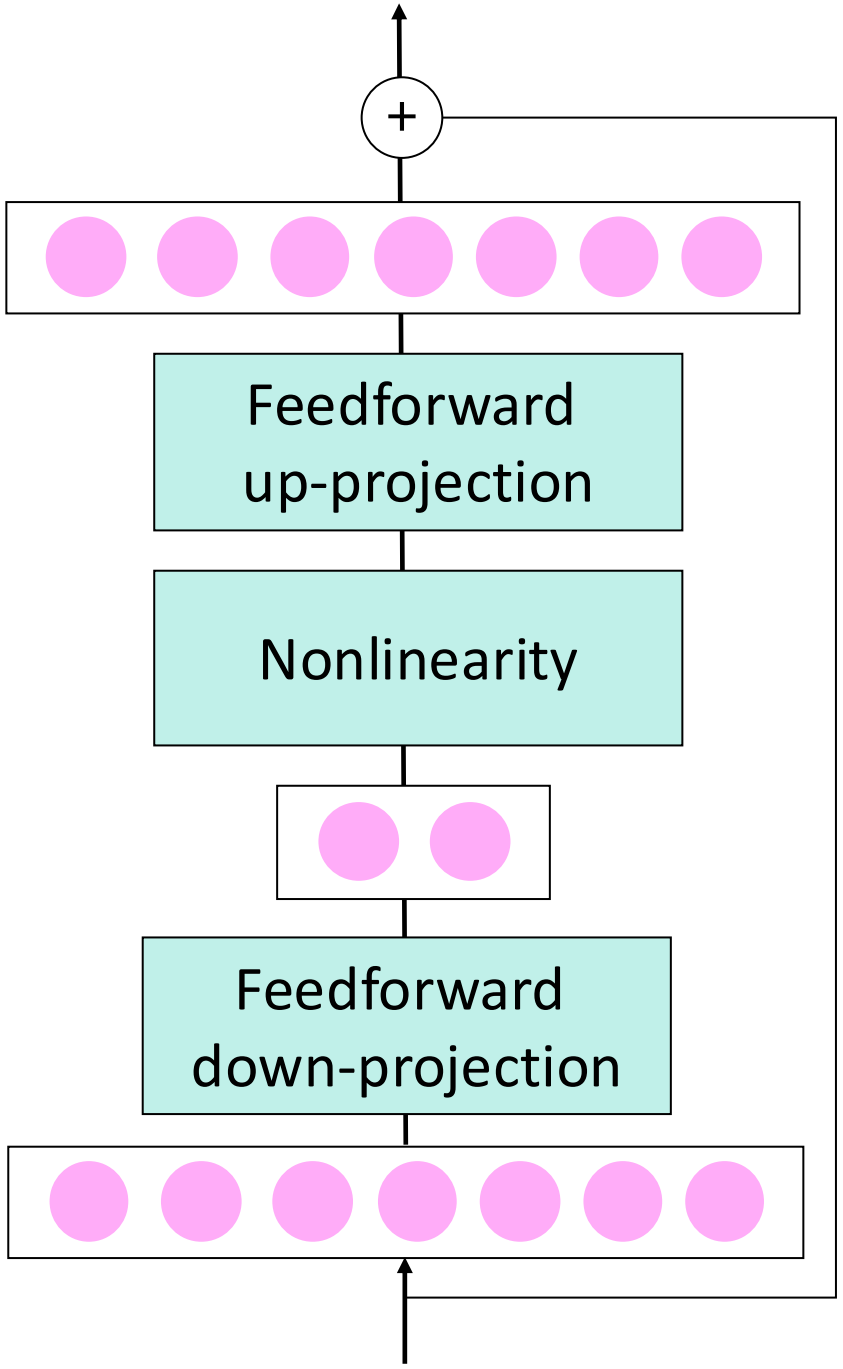
Adapter (Houlsby et al. 2019)

- Insert a new function f_ϕ between layers of a pre-trained model to adapt to a downstream task --- known as "adapters"

- An **adapter** in a Transformer layer consists of:

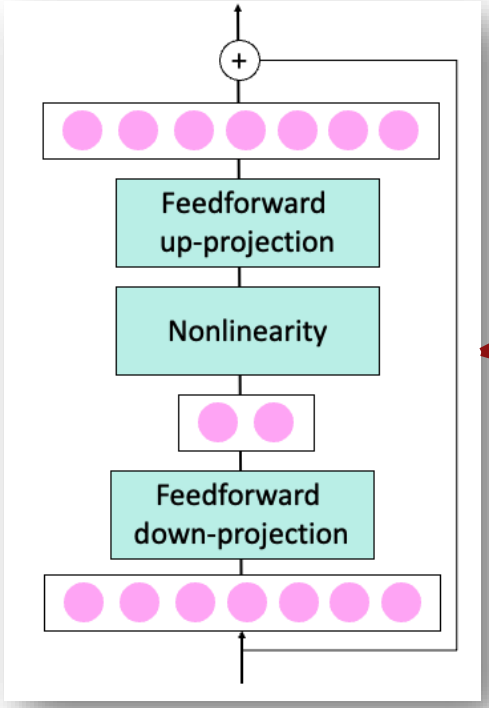
  - A feed-forward down-projection $W^D \in R^{k \times d}$
  - A feed-forward up-projection $W^U \in R^{d \times k}$

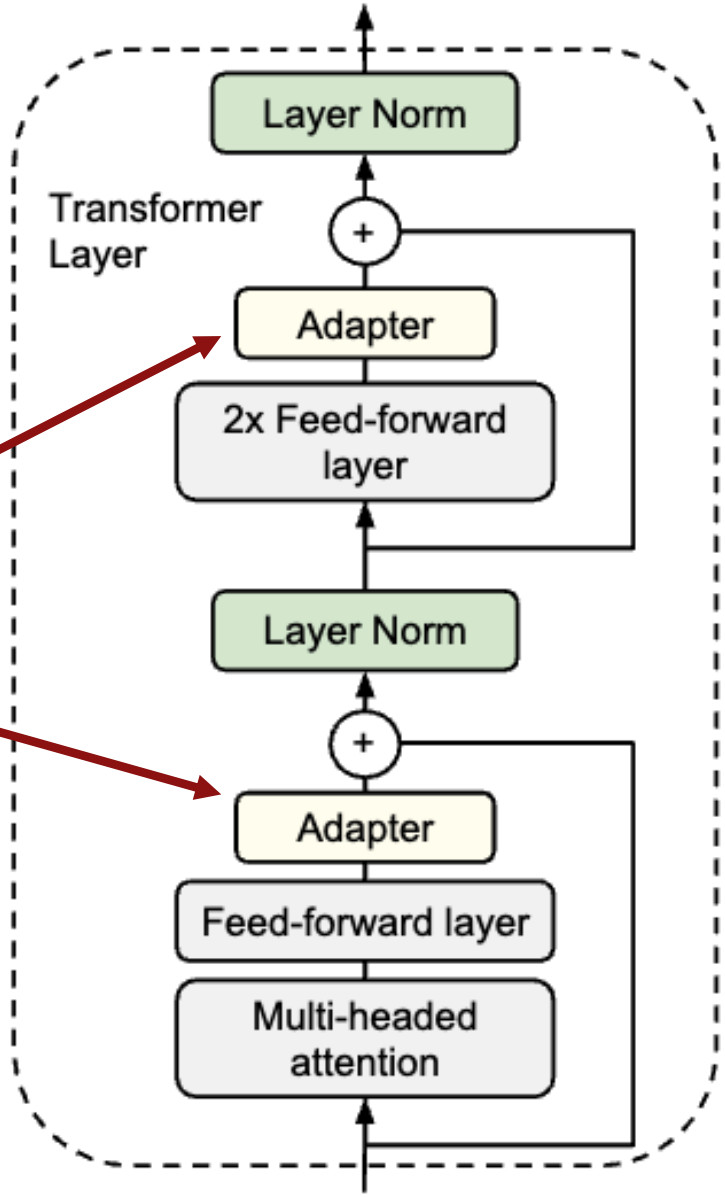
  - $f_\phi(\boldsymbol{x}) = W^U(\sigma(W^D \boldsymbol{x}))$



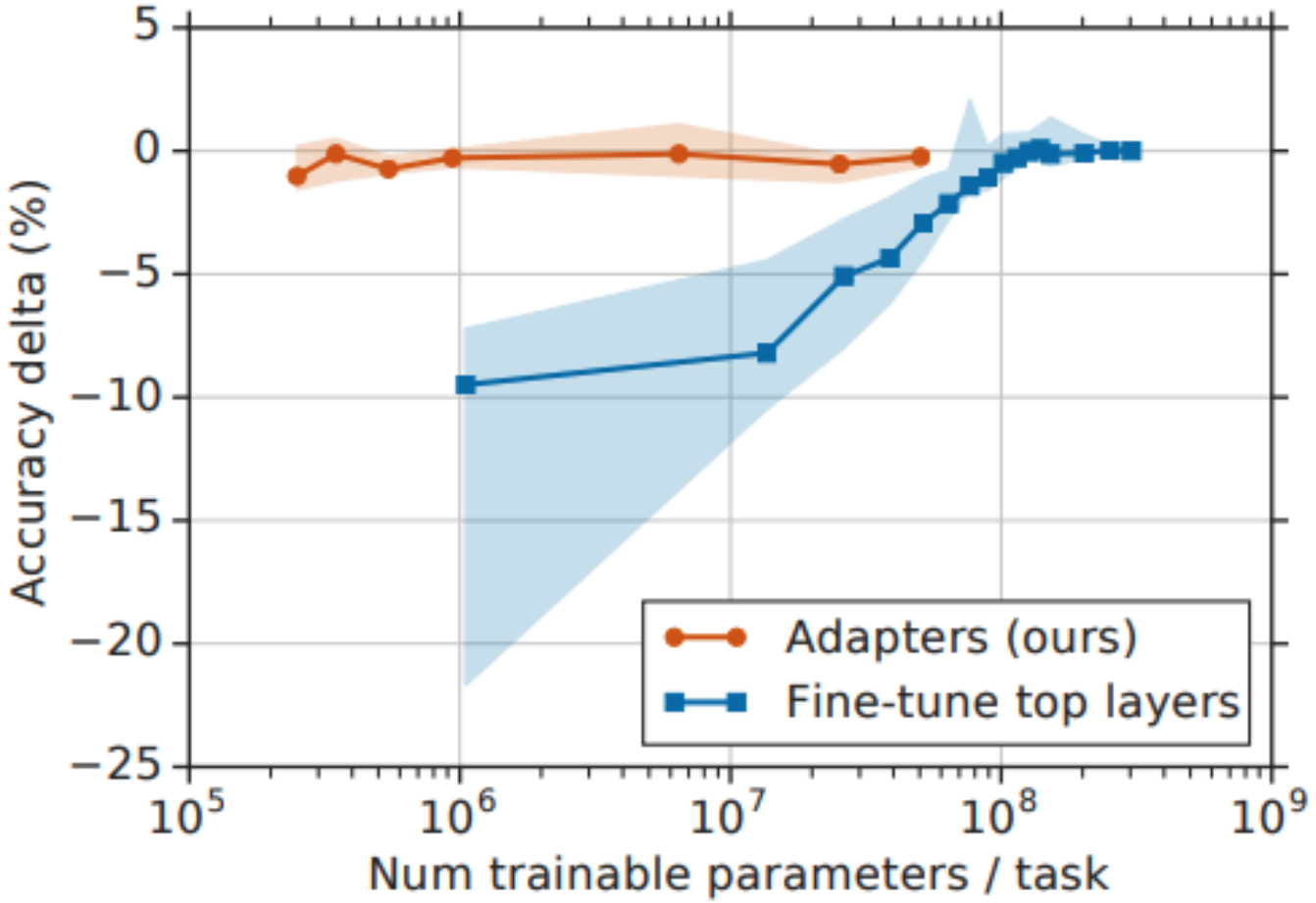
Adapter (Houlsby et al. 2019)

- The adapter is usually placed after the multi-head attention and/or after the feed-forward layer
- Most approaches have used this bottleneck design with linear layers





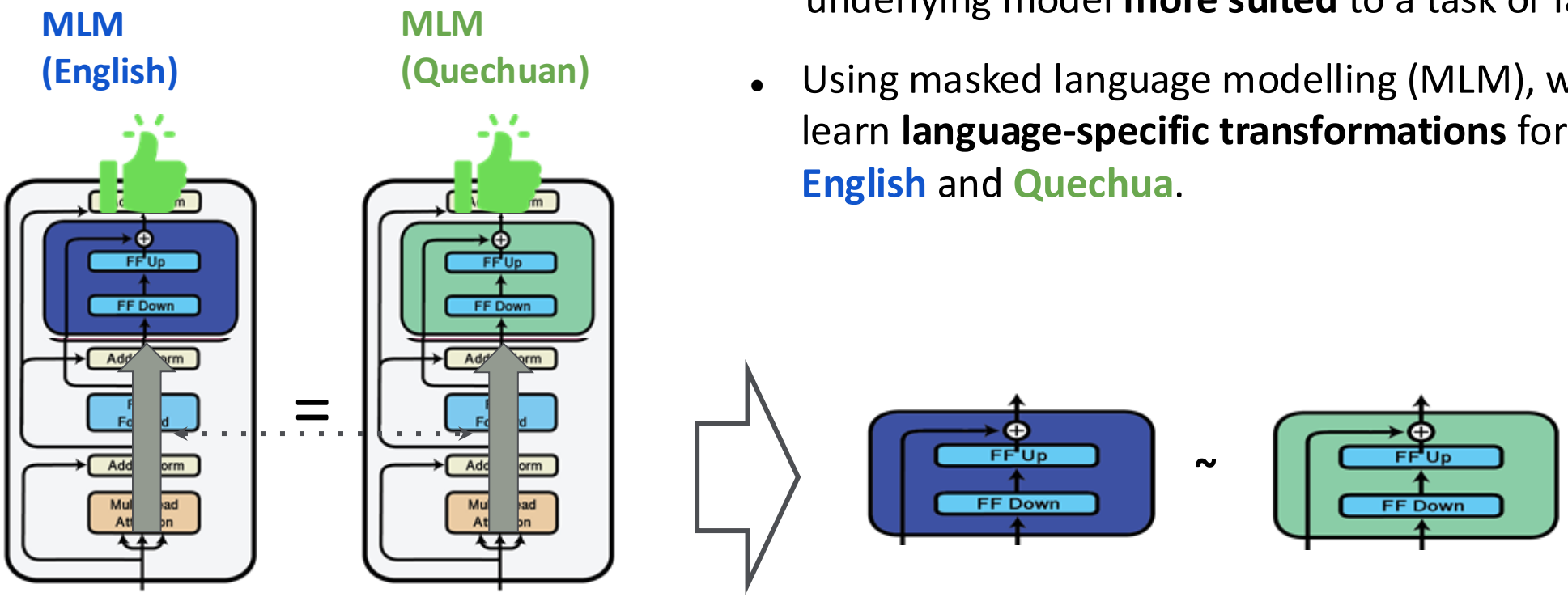
Trade-off btw accuracy and # of trained task specific parameters



The curves show the 20th, 50th, and 80th performance percentiles across nine tasks from the GLUE benchmark.

Adapter based tuning attains a similar performance to full finetuning with two orders of magnitude fewer trained parameters

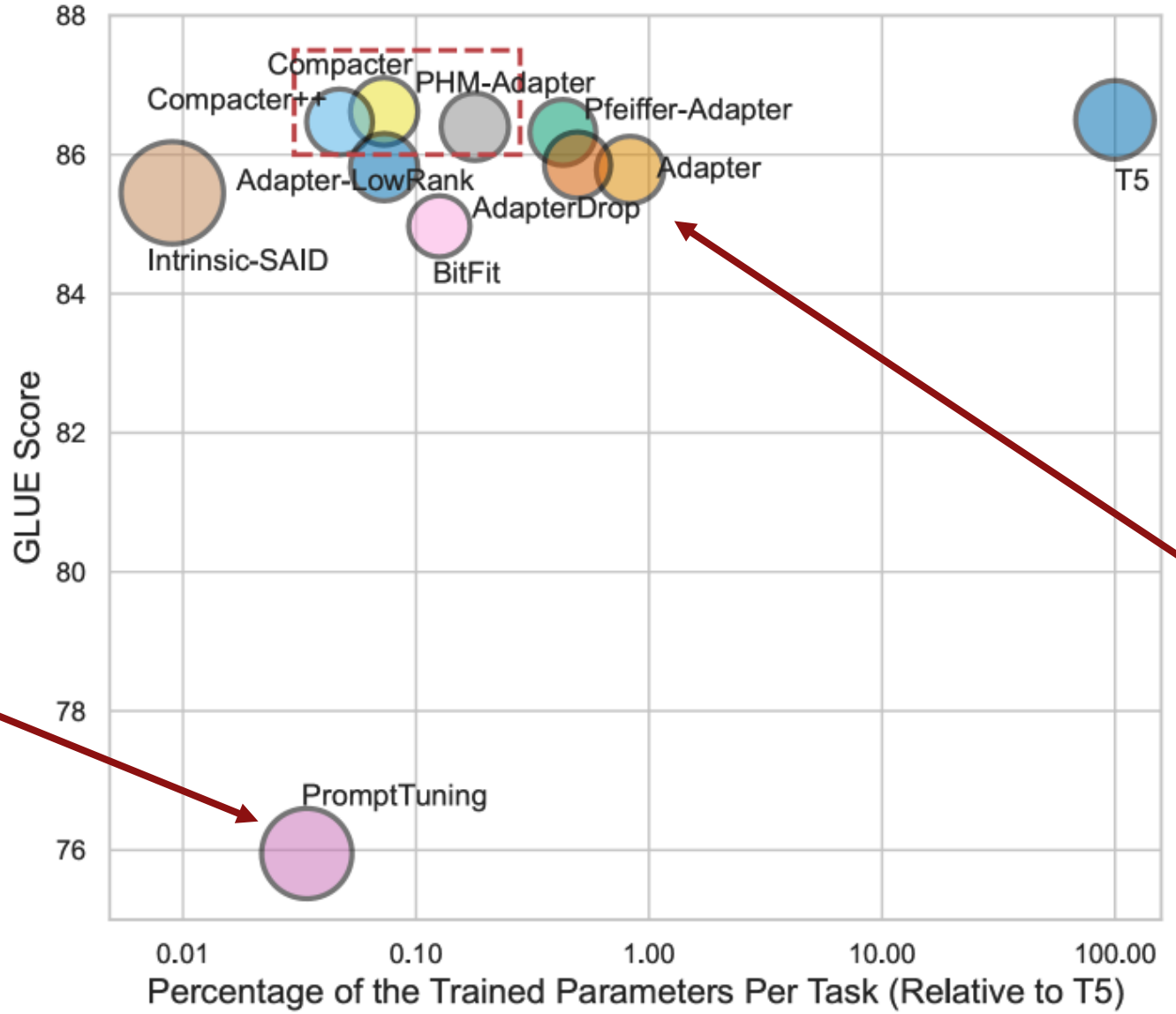
Language adapters? Task knowledge ~= language knowledge



- Adapters **learn transformations** that make the underlying model **more suited** to a task or language.
- Using masked language modelling (MLM), we can learn **language-specific transformations** for e.g. **English** and **Quechua**.

Performance comparison

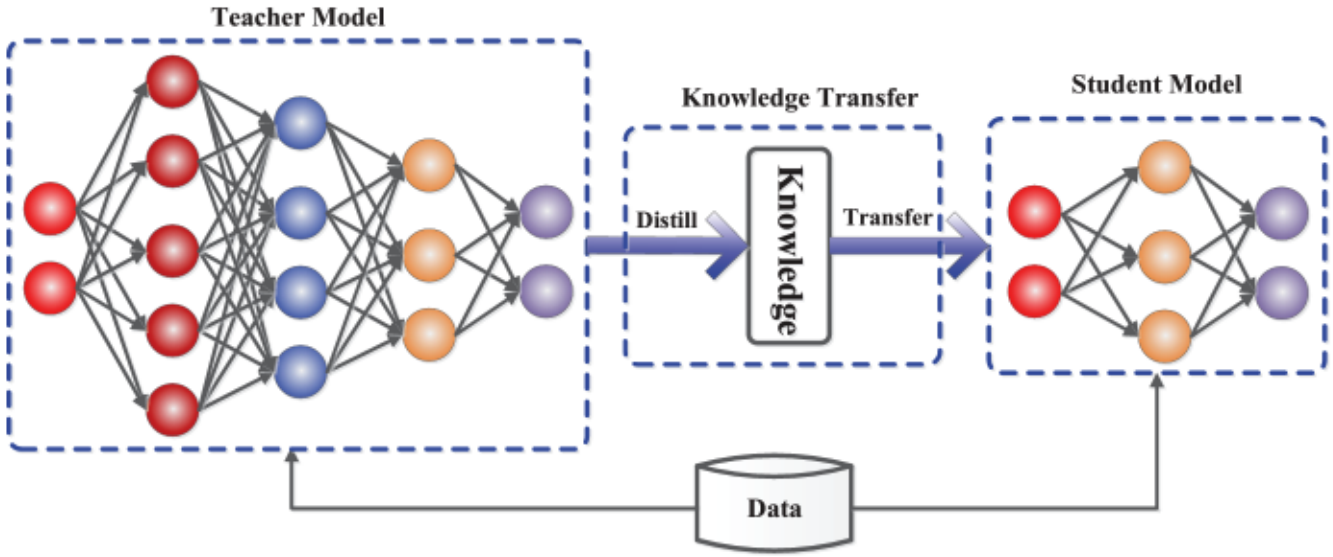
Prompt tuning underperforms the other methods due to limited capacity



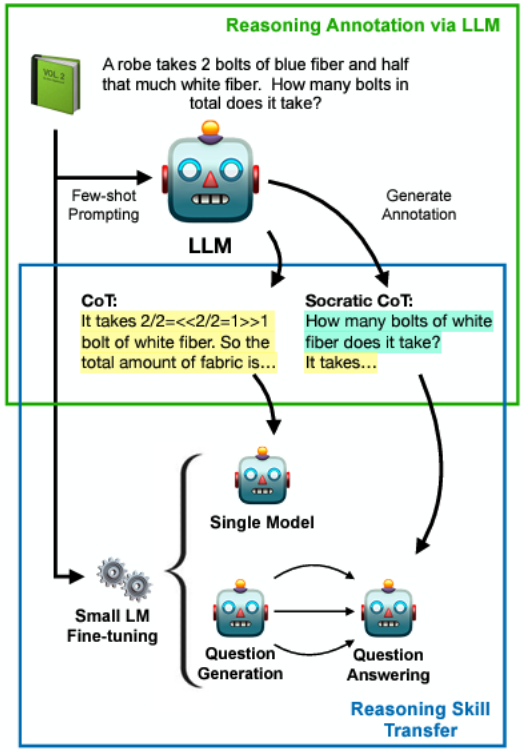
Adapter achieves better performance but add more parameters

Other variants of (efficient) adaptation

- **Knowledge distillation** to obtain smaller models



The generic teacher-student framework for knowledge distillation (Gou et al.,)



Shridhar et al., 2023

- **Also check out**: Gist tokens (Wu et al., 2024), ReFT(Wu et al, 2024), etc

1. Question Answering

Reading Comprehension

The first recorded travels by Europeans to China and back date from this time. The most famous traveler of the period was the Venetian Marco Polo, whose account of his trip to "Cambaluc," the capital of the Great Khan, and of life there astounded the people of Europe. The account of his travels, Il milione (or, The Million, known in English as the Travels of Marco Polo), appeared about the year 1299. Some argue over the accuracy of Marco Polo's accounts due to the lack of mentioning the Great Wall of China, tea houses, which would have been a prominent sight since Europeans had yet to adopt a tea culture, as well the practice of foot binding by the women in capital of the Great Khan. Some suggest that Marco Polo acquired much of his knowledge through contact with Persian traders since many of the places he named were in Persian.

How did some suspect that Polo learned about China instead of by actually visiting it?

Answer: through contact with Persian traders

How many slices of pizza are there?
Is this a vegetarian pizza?

how many slices of pizza are there? is this a vegetarian pizza?

ChatGPT

There are 8 slices of pizza in the image. As for whether it's a vegetarian pizza, it appears to have toppings that could be vegetables, such as pieces that look like diced tomatoes and possibly green peppers. However, without a clearer image or a list of ingredients, it's impossible to be certain if there are no meat products on the pizza. Vegetarian pizzas typically do not include meat, but they can include a variety of vegetable toppings like onions, peppers, mushrooms, olives, and more. If there is any meat or meat-based toppings visible, then it would not be a vegetarian pizza.

(Rajpurkar et al, 2016): SQuAD: 100,000+ Questions for Machine Comprehension of Text

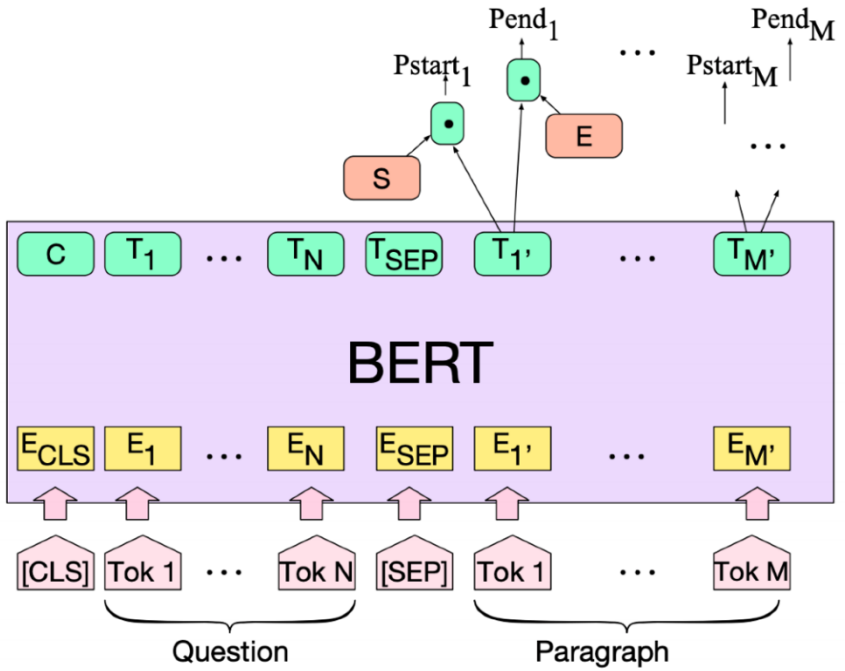
Conventional methods for reading comprehension

Feature-based

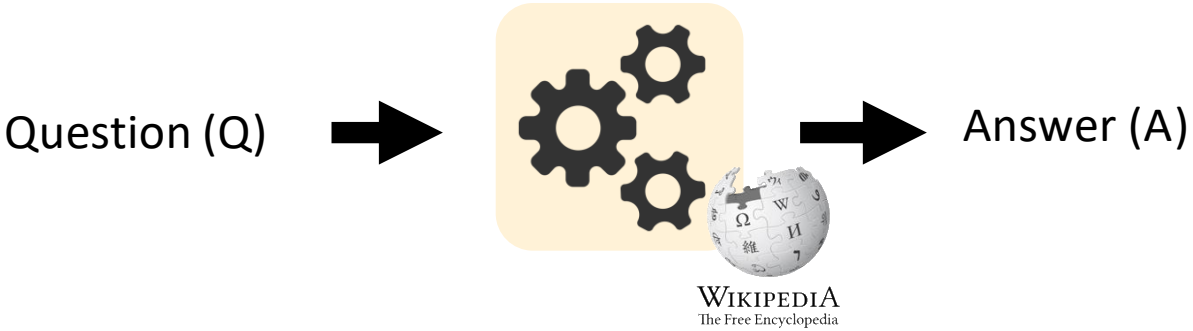
- Generate a list of candidate answers $(a_1, a_2, \ldots, a_M)$
- Define a feature vector $\phi(p, q, a_i) \in R^d$: Word/bigram features; Parse tree matches
- Apply a multi-class logistic regression model

Neural approach:

- Problem formulation
  - Input: $C = (c_1, c_2, \ldots, c_N), Q = (q_1, q_2, \ldots, q_M)$
  - Output: 1≤ start ≤ end ≤N
  - N~100, M ~15



Open-domain question answering

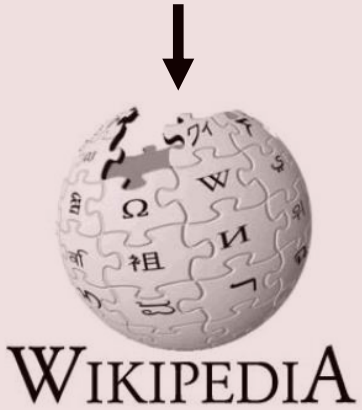


- Different from reading comprehension, we don't assume a given passage.
- Instead, we only have access to a large collection of documents (e.g., Wikipedia). We don't know where the answer is located, and the goal is to return the answer for any open-domain questions.
- Much more challenging and a more practical problem!

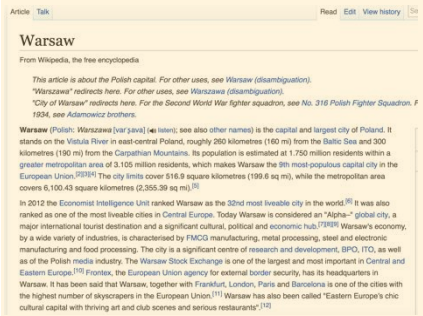
*In contrast to **closed-domain** systems that deal with questions under a specific domain (medicine, technical support).*

Retrieval augmentation

How many of Warsaw's inhabitants spoke Polish in 1933?



Document Retriever


Warsaw

From Wikipedia, the free encyclopedia

Document Reader

833,500

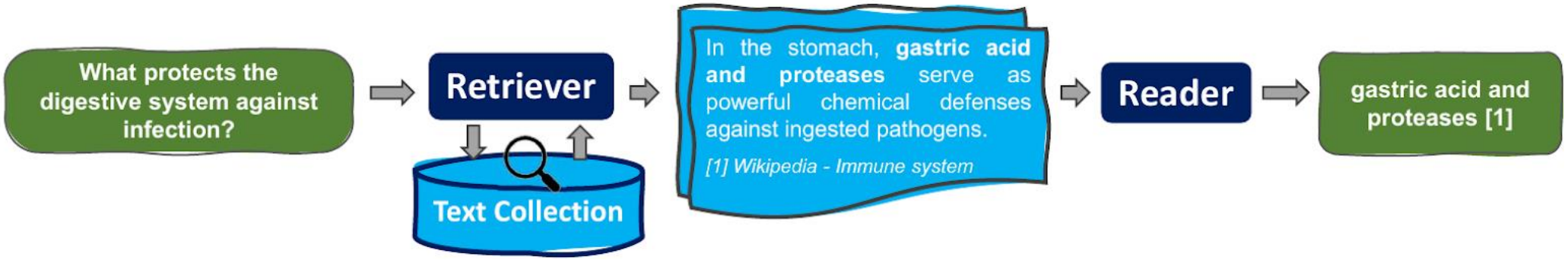
https://github.com/facebookresearch/DrQA

Chen et al., 2017. Reading Wikipedia to Answer Open-domain Questions

Using retrieval to overcome LMs' shortcomings

- Instead of asking the LM to memorize everything, can we provide the LM with relevant and useful content just-in-time?

- Retrieval / search is a common mechanism for identifying such relevant information.
  - **Dynamic**: it's easy to update / add documents to your retrieval system
  - **Interpretable**: LM can generate pointers to retrieved documents that support human verification of its generations (citations)



Retriever-Reader framework

- Input: a large collection of documents $\mathcal{D} = D_1, D_2, \ldots, D_N$ and Q
- Output: an answer string A

Retriever: $f(\mathcal{D}, Q) \longrightarrow P_1, \ldots, P_K$

Reader: $g(Q, \{P_1, \ldots, P_K\}) \longrightarrow A$

K is pre-defined (e.g., 100)

A reading comprehension problem!

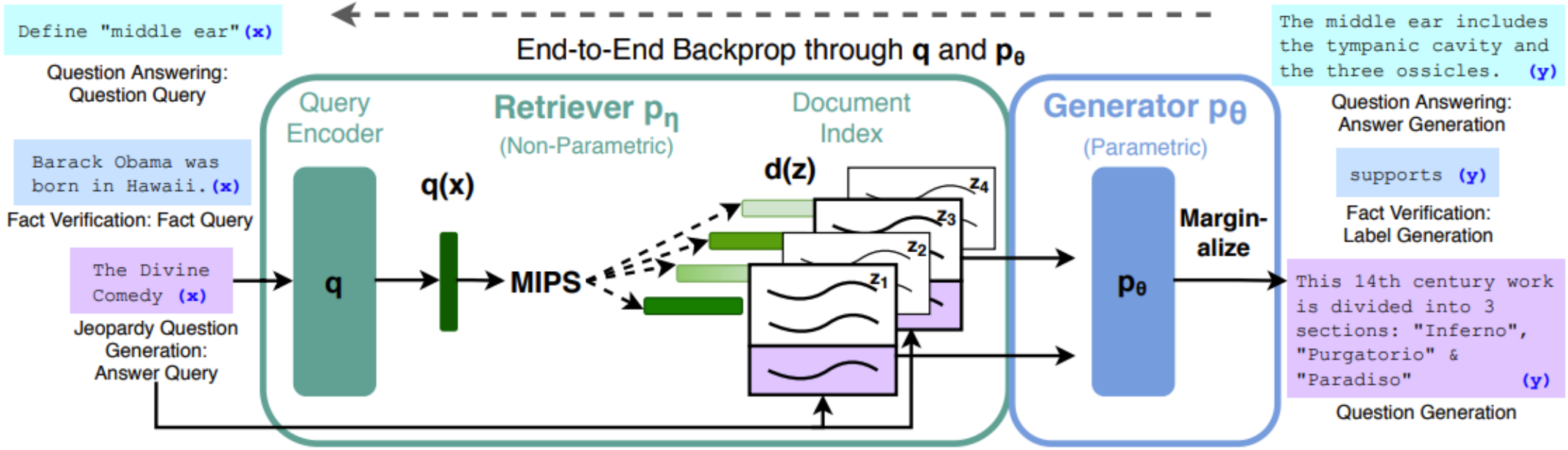
Retriever = A standard TF-IDF information-retrieval sparse model (a fixed module)

Reader = a neural reading comprehension model

- Could be Trained on SQuAD and other distantly-supervised QA datasets
- Or a zero-shot LLM (ChatGPT etc)

RAG and it's open problems

Retrieval Augmented Generation (RAG) is very powerful!



See also works like REALM, DPR, ORQA etc..

[Lewis et al., 2021]

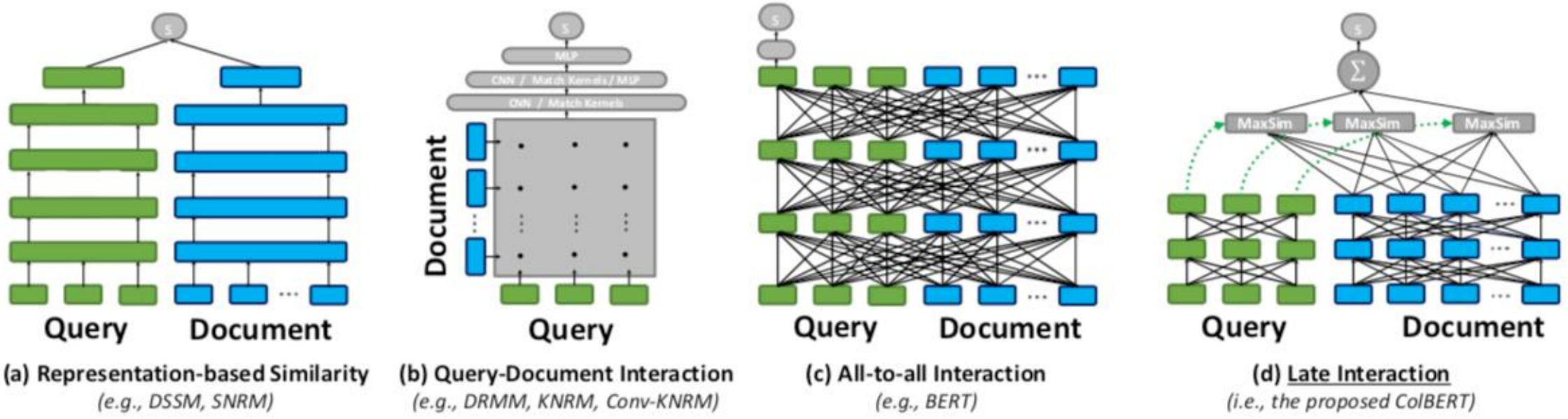
Different types of retrievers

How should we retrieve relevant passages for our retrieval system?

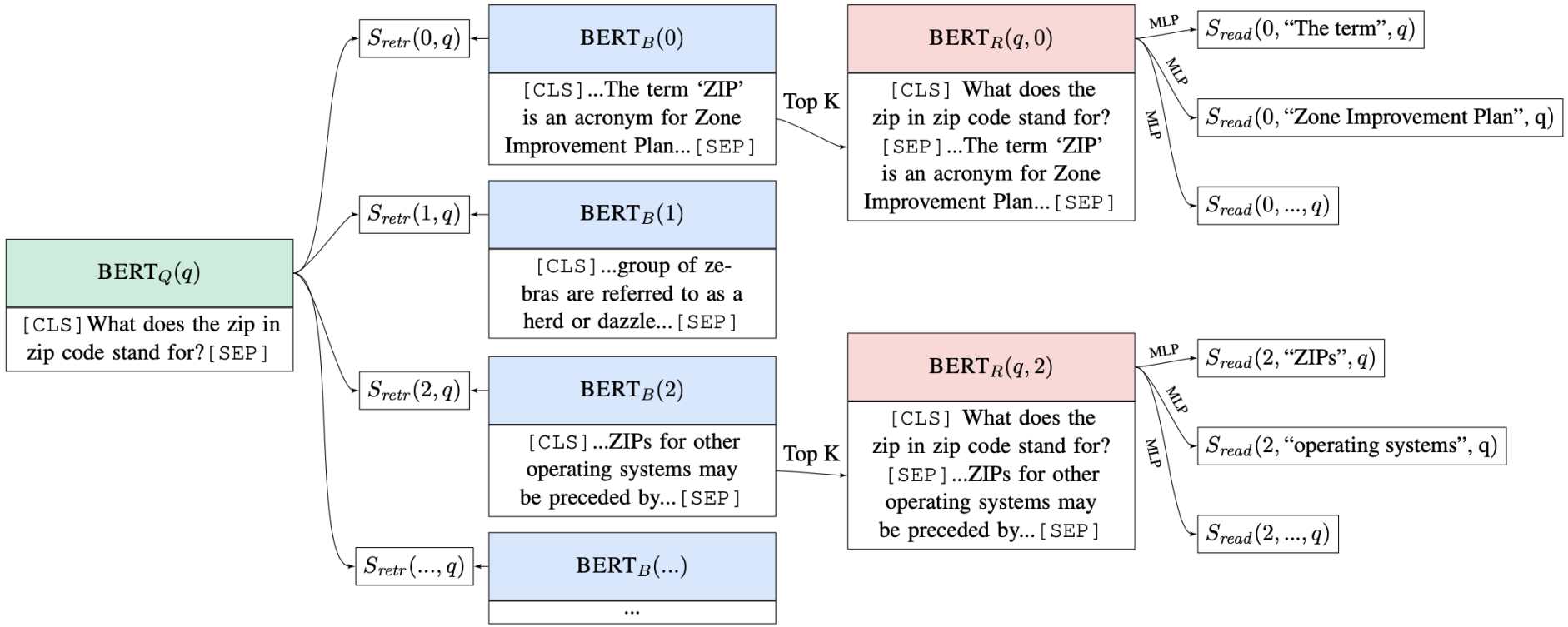
- Word-overlap (BM25)
- Vector retrieval (DPR, Sentence vectors)
- Other, neural systems (ColBERT)

Fast vs slow retrievers

- Fastest: computing similarities (or word overlap) on pre-computed vectors
- Slowest: using a LM to compute similarities
- Other hybrid (pre-computing the ‘document index’) e.g. in ColBERT



We can train the retriever! Joint training of reader and retriever

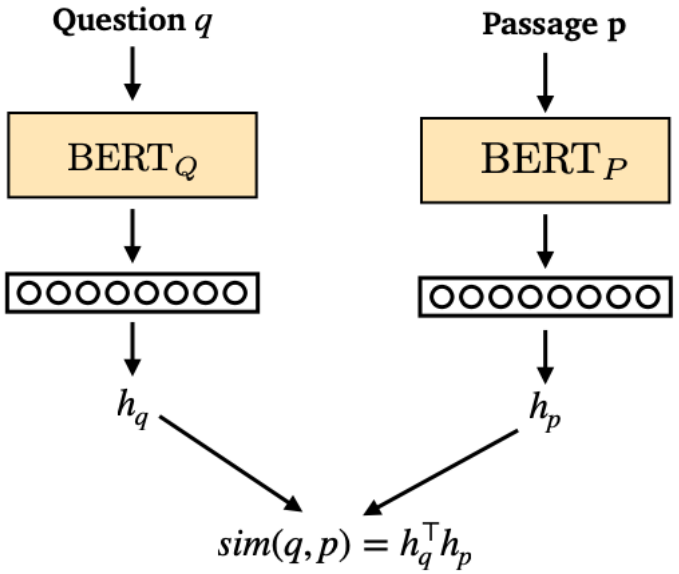


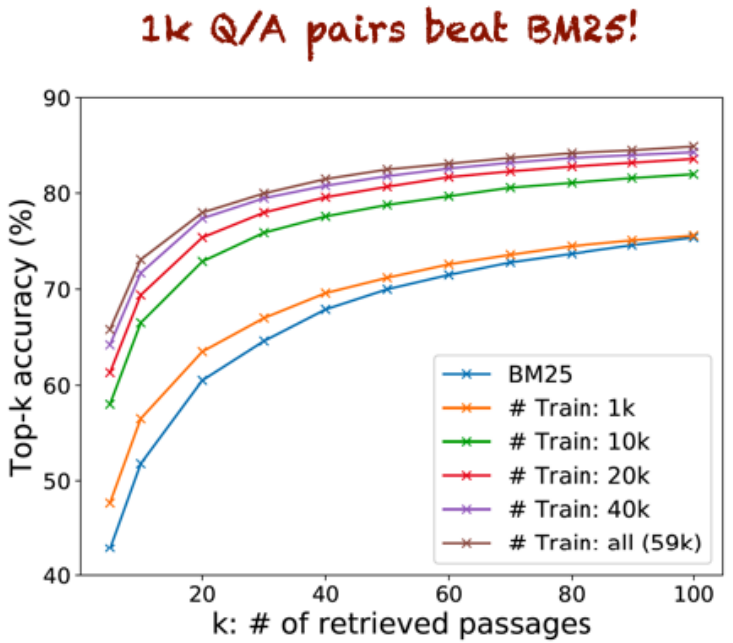
- Each text passage can be encoded as a vector using BERT and the retriever score can be measured as the dot product between the question representation and passage representation.
- However, it is not easy to model as there are a huge number of passages (e.g., 21M in English Wikipedia)

Lee et al., 2019. Latent Retrieval for Weakly Supervised Open Domain Question Answering

We can train the retriever!

- Dense passage retrieval (DPR)
- We can also just train the retriever using question-answer pairs!



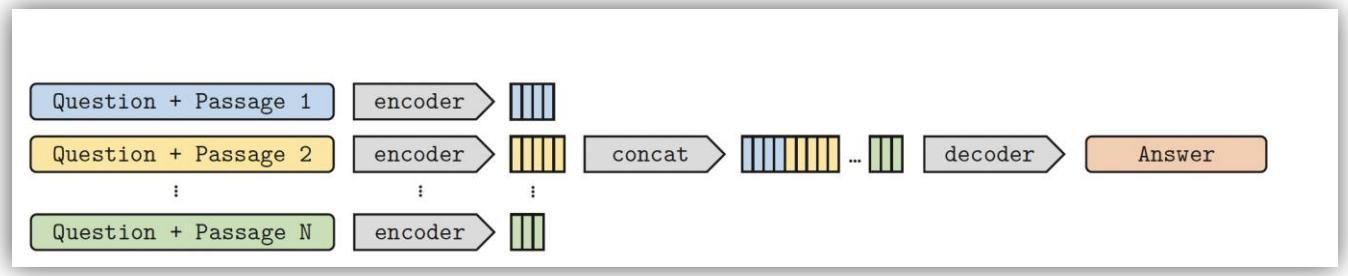


- Trainable retriever (using BERT) largely outperforms traditional IR retrieval models

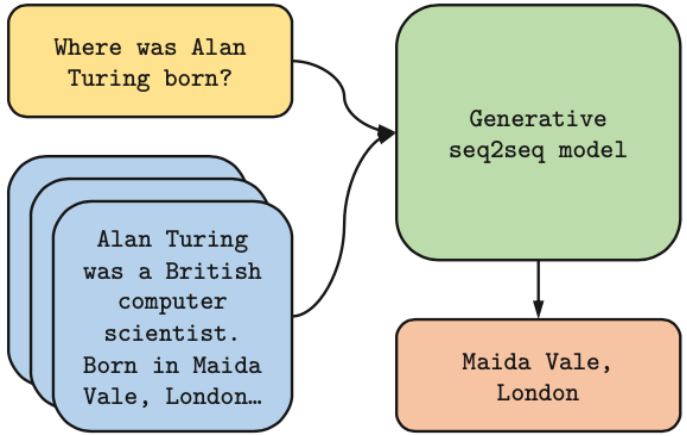
Karpukhin et al., 2020. Dense Passage Retrieval for Open-Domain Question Answering

Deep retrieval + generative models

- Recent work shows that it is beneficial to generate answers instead of to extract answers.



Fusion-in-decoder (FID)=DPR+T5



Model	NaturalQuestions	TriviaQA	
ORQA (Lee et al., 2019)	31.3	45.1	-
REALM (Guu et al., 2020)	38.2	-	-
DPR (Karpukhin et al., 2020)	41.5	57.9	-
SpanSeqGen (Min et al., 2020)	42.5	-	-
RAG (Lewis et al., 2020)	44.5	56.1	68.0
T5 (Roberts et al., 2020)	36.6	-	60.5
GPT-3 few shot (Brown et al., 2020)	29.9	-	71.2
Fusion-in-Decoder (base)	48.2	65.0	77.1
Fusion-in-Decoder (large)	**51.4**	**67.6**	**80.1**

Izacard and Grave 2020. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering

Problem: How many documents can we use?

The retriever is key – if we have to use only one document, then we have to get that right.

Why not get lots of documents and pass it to the LM?

Input Context

```
Write a high-quality answer for the given question using only the provided search
results (some of which might be irrelevant).

Document [1](Title: Asian Americans in science and technology) Prize in physics for
discovery of the subatomic particle J/ψ. Subrahmanyan Chandrasekhar shared...
Document [2](Title: List of Nobel laureates in Physics) The first Nobel Prize in
Physics was awarded in 1901 to Wilhelm Conrad Röntgen, of Germany, who received...
Document [3](Title: Scientist) and pursued through a unique method, was essentially
in place. Ramón y Cajal won the Nobel Prize in 1906 for his remarkable...

Question: who got the first nobel prize in physics
Answer:
```

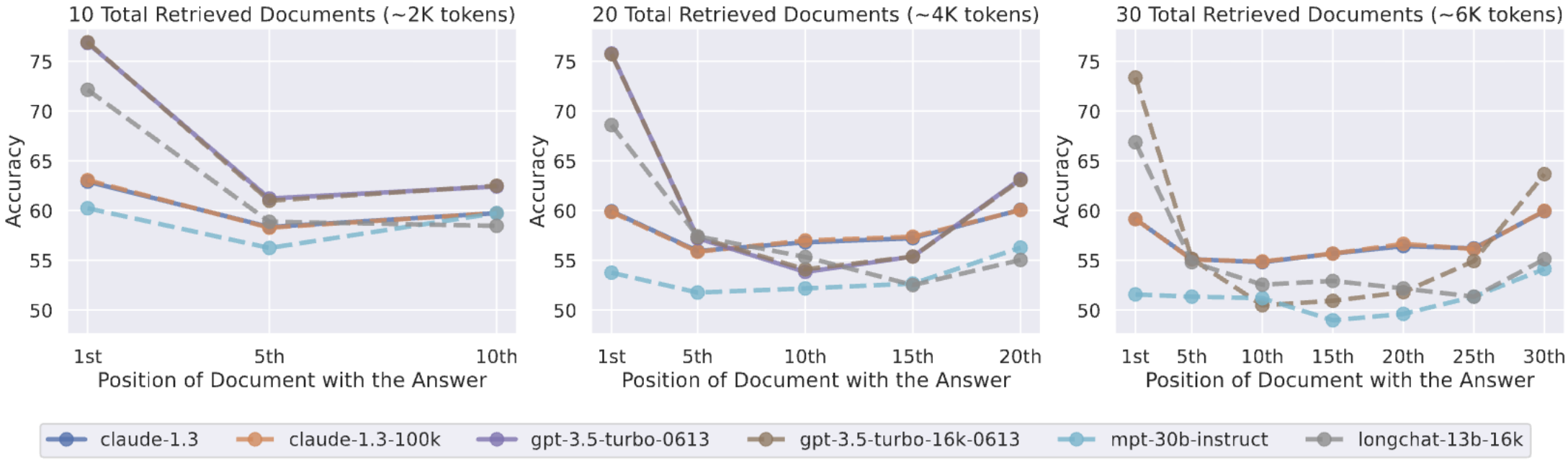
Desired Answer

```
Wilhelm Conrad Röntgen
```

LM's can't pay attention to the entire context [Liu+ 2023]

The long-context problem bites you – LMs do not pay attention to its context well!

Setup: 1 relevant document, all others irrelevant



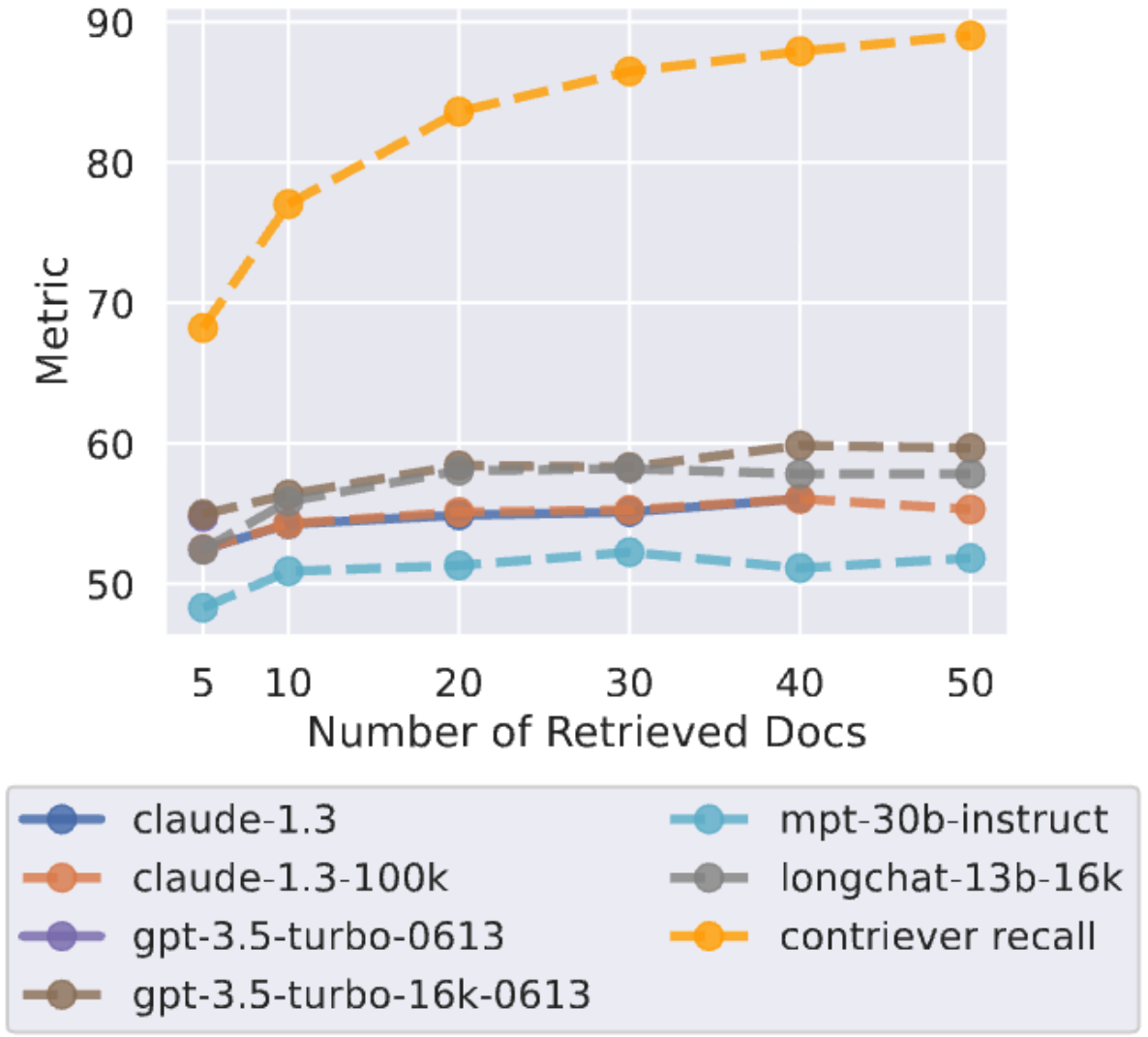
Best Closed-Book performance: GPT-3.5-Turbo, ~56%

Best Oracle (only feed in relevant doc) performance: GPT-3.5-Turbo, ~88.5%

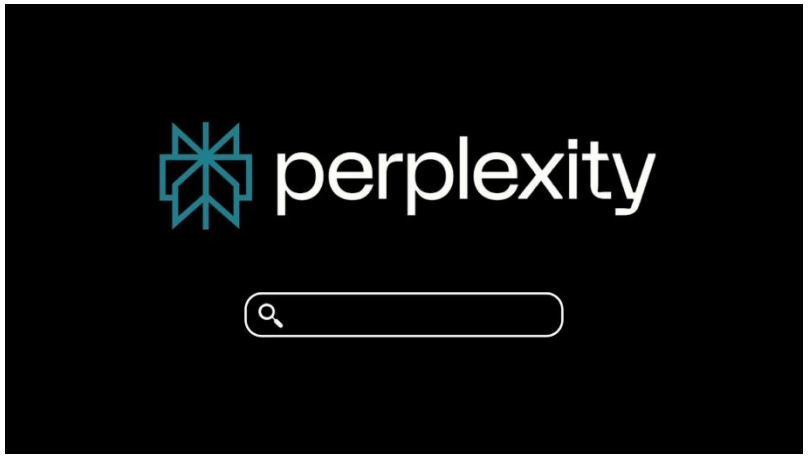
In practice: LMs cant use many documents

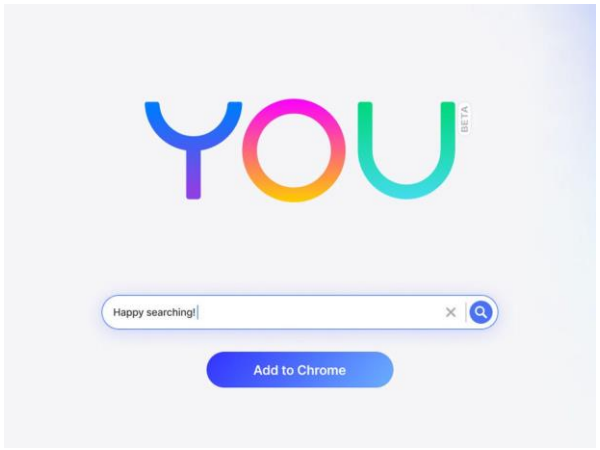
Retriever performance (yellow) rises slowly to 90% recall

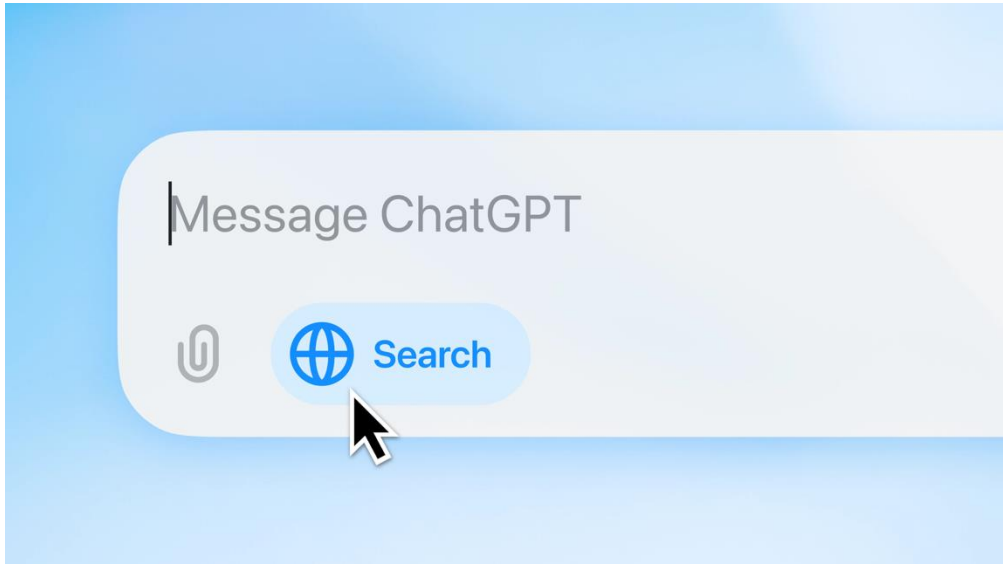
RAG performance (other lines) saturate very quickly – after 10-20 documents.

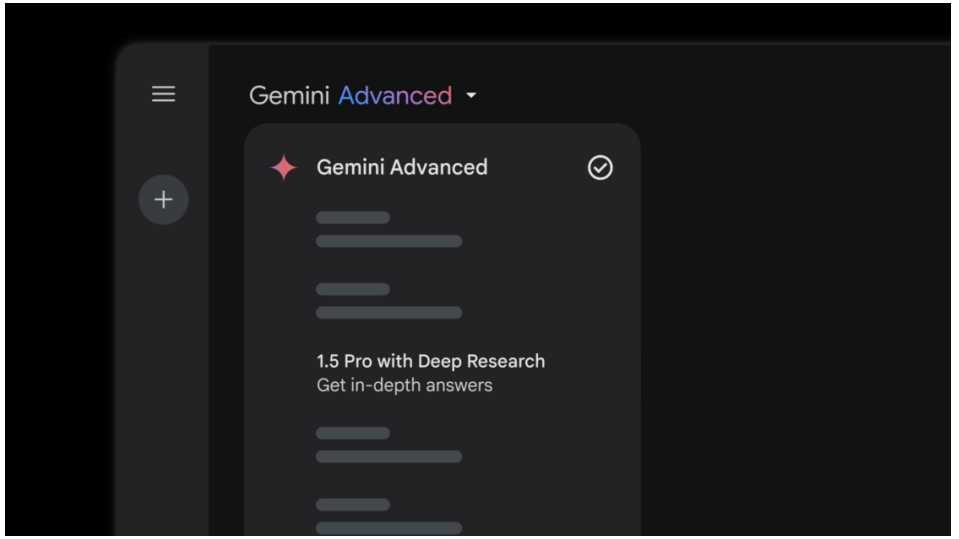


LLMs with web search (systems that search and cite the web)









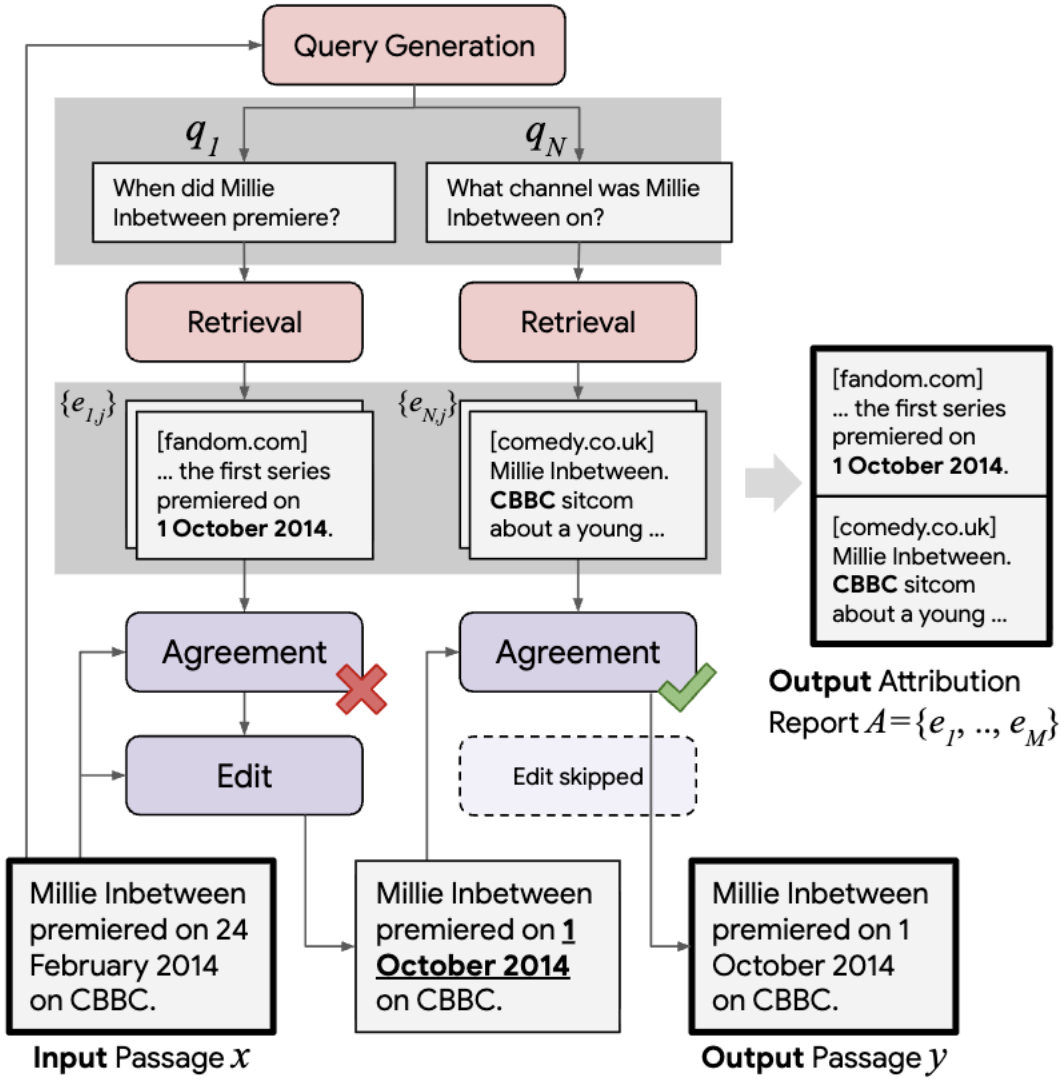
RARR: research and revise what LLMs say using LLMs [Guo+2023]

Uses QA to check for consistency between sources and responses

Returns citations for each of these items using a LM.

Revises by detecting disagreement between the text and citations, then runs an edit model to revise the text if needed

Evidence snippets are selected to form an attribution report

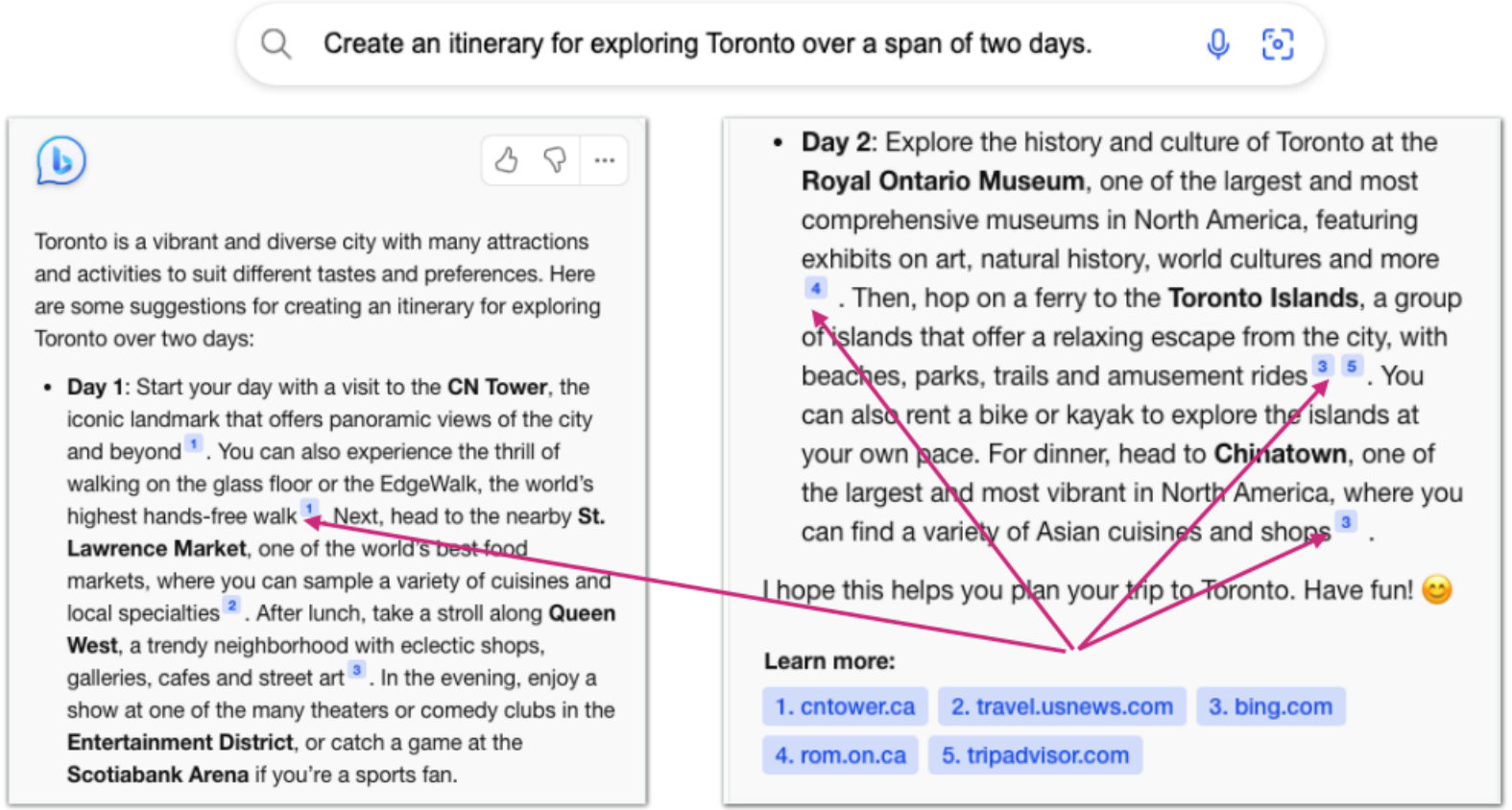


LLMs citation and hallucinations

Unique benefit of RAG: citing the source

These citations themselves are generated by LLMs

So the citations could *also be hallucinated* 🤷‍♀️ How often does that happen?



LLMs citation and hallucinations

Unique benefit of RAG: citing the source

These citations themselves are generated by LLMs

So the citations could *also be hallucinated* 🤷‍♀️ How often does that happen?

Outputs are easy to read / appear useful to rater (1-5 scale)

Perceived Utility (↑)	Average Over All Queries
Bing Chat	4.34
NeevaAI	4.48
perplexity.ai	4.56
YouChat	4.62
Average	4.50

Fluency (↑)	Average Over All Queries
Bing Chat	4.40
NeevaAI	4.43
perplexity.ai	4.51
YouChat	4.59
Average	4.48

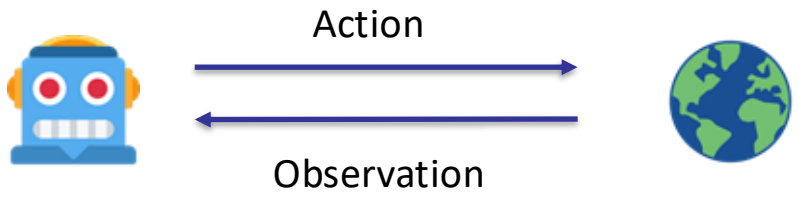
But precision and recall are both low..

Citation Precision (%; ↑)	Average Over All Queries
Bing Chat	89.5
NeevaAI	72.0
perplexity.ai	72.7
YouChat	63.6
Average	74.5

Citation Recall (%; ↑)	Average Over All Queries
Bing Chat	58.7
NeevaAI	67.6
perplexity.ai	68.7
YouChat	11.1
Average	51.5

2. Language agents: from words to action

- LLMs predict textual output for a given input; while agents perform actions based on observations from the environment/world



- Lots of debate on what is an agent and what is not an agent
- Same with the definition of “agentic”
- Here, we focus on key components of LLM empowered systems or language agents

Many slides credit to:
Language Agents: Foundations, Prospects, and Risks. Yu Su, Diyi Yang, Shunyu Yao, Tao Yu. EMNLP 2024 Tutorial.

Language agents: key components

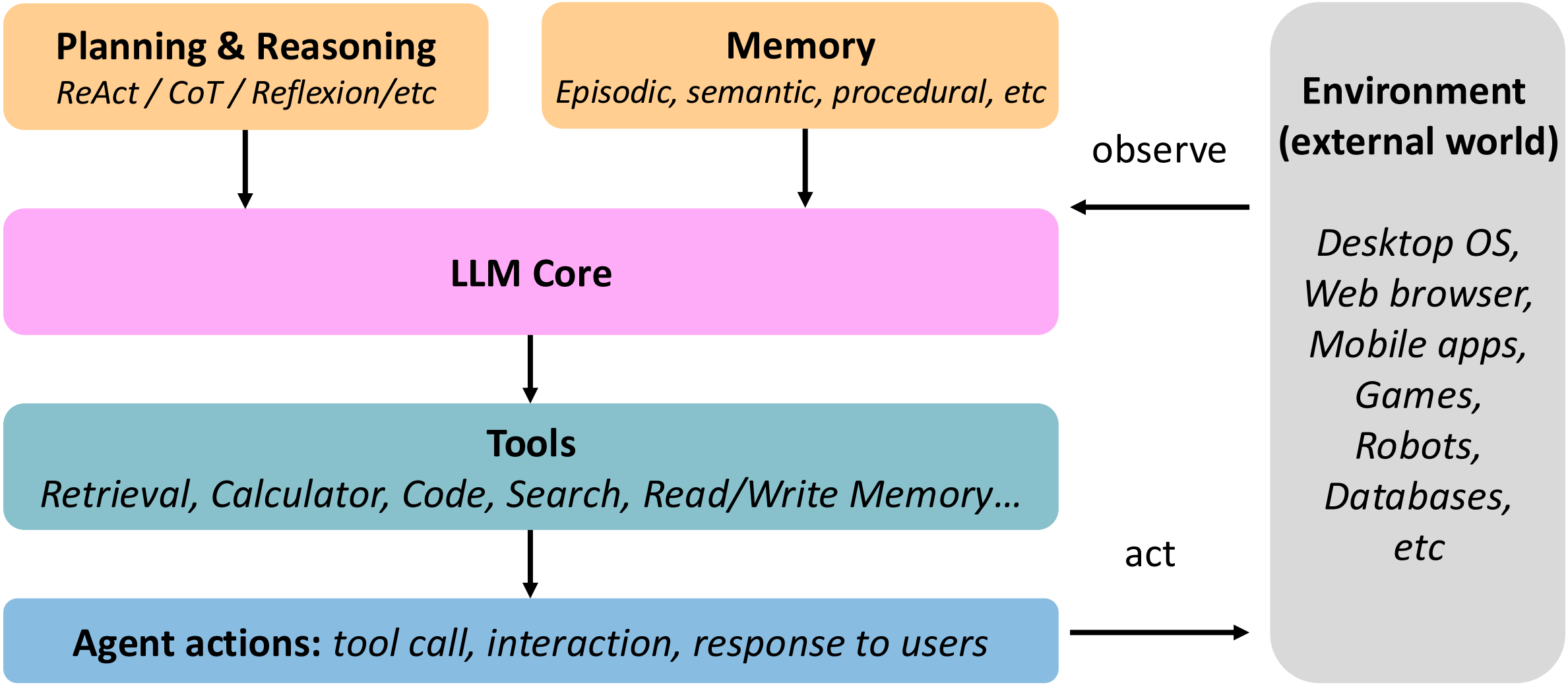
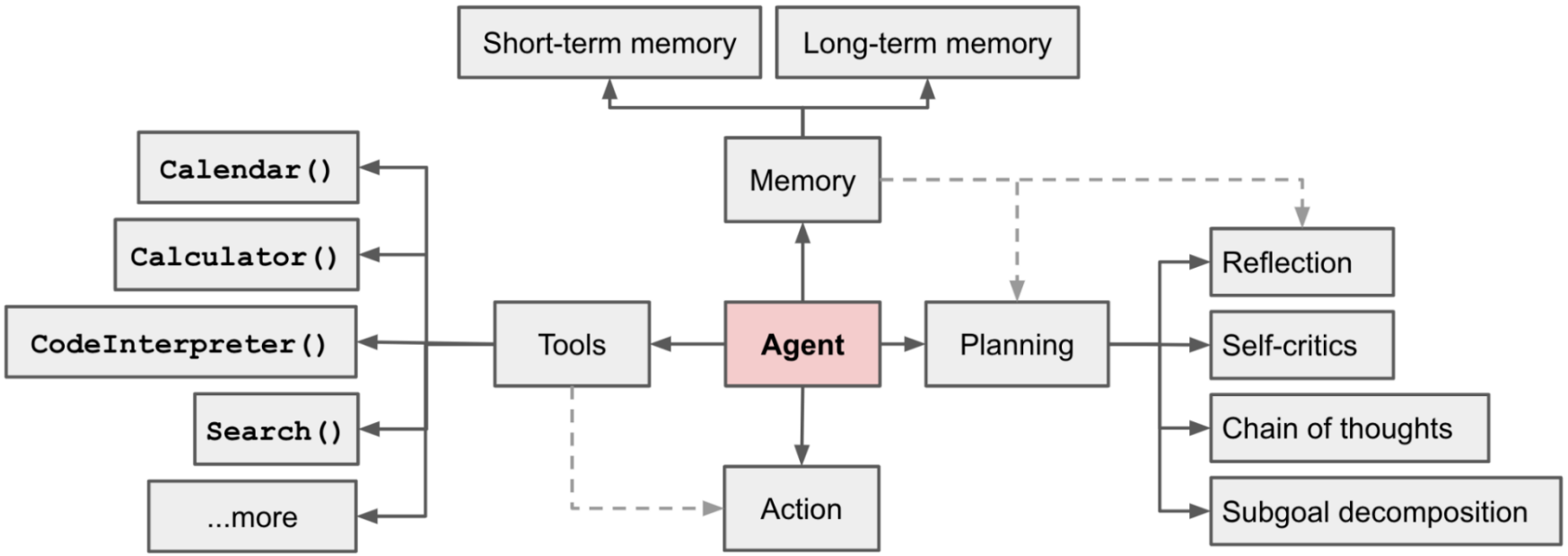
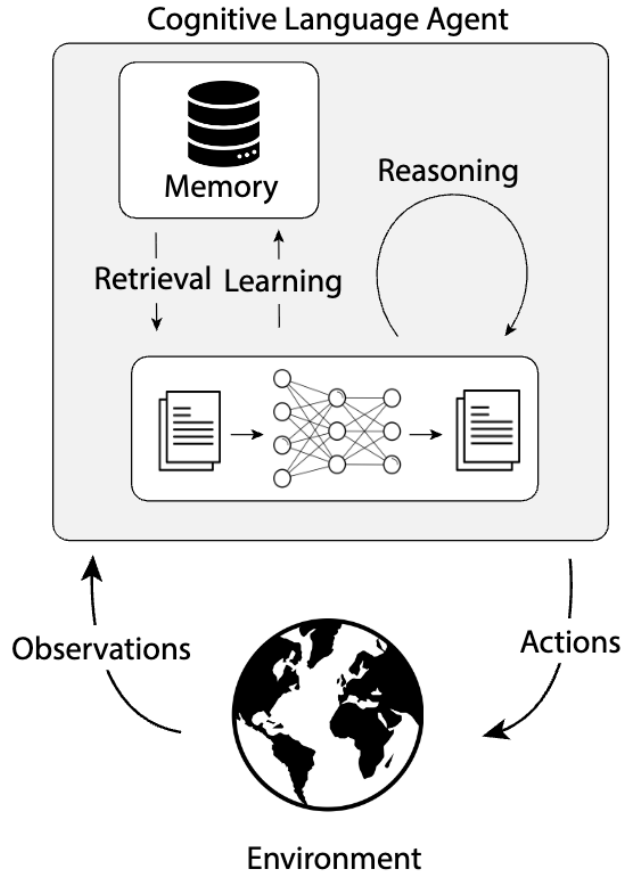


Illustration purpose only; not meant to be the exact process of agent development

Other types of common components of language agents

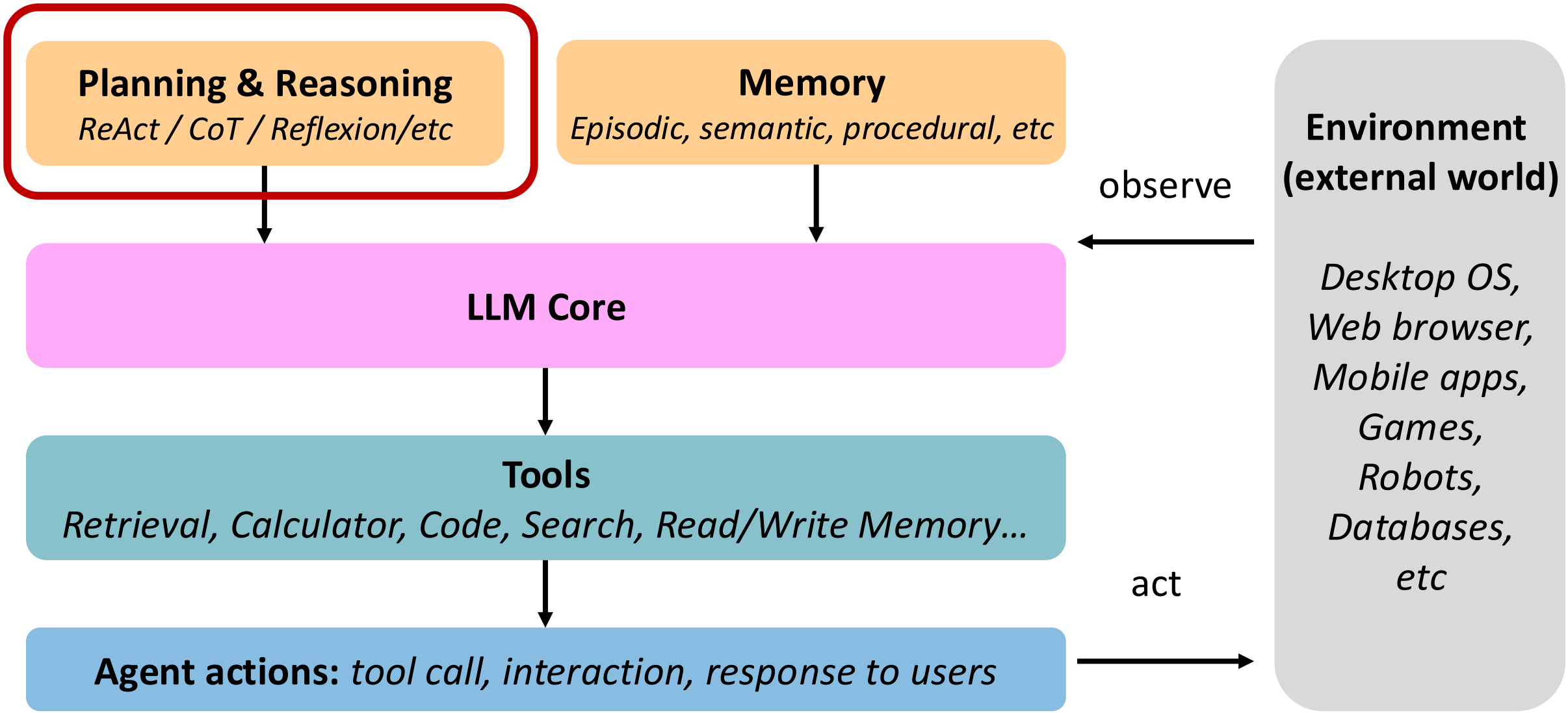


https://lilianweng.github.io/posts/2023-06-23-agent/



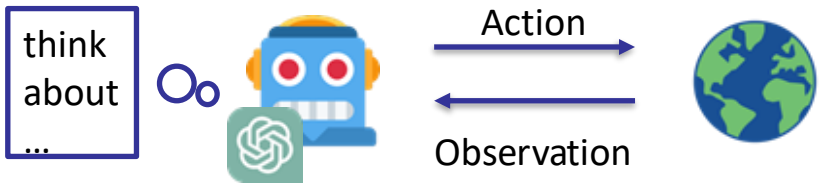
Cognitive Language Agents (Sumers and Yao, 2024)

3. Language agents: Reasoning



Reasoning

- For humans: various mental processes
- For LLMs: intermediate generation that imitate various (but not all) human mental processes
- For agents: internal actions



PERCEPTION | INTUITION SYSTEM 1 | REASONING SYSTEM 1

Math Word Problems (free response)

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Math Word Problems (multiple choice)

Q: How many keystrokes are needed to type the numbers from 1 to 500? Answer Choices: (a) 1156 (b) 1392 (c) 1480 (d) 1562 (e) 1788

A: There are 9 one-digit numbers from 1 to 9. There are 90 two-digit numbers from 10 to 99. There are 401 three-digit numbers from 100 to 500. 9 + 90(2) + 401(3) = 1392. The answer is (b).

StrategyQA

Q: Yes or no: Would a pear sink in water?

A: The density of a pear is about 0.6 g/cm^3, which is less than water. Thus, a pear would float. So the answer is no.

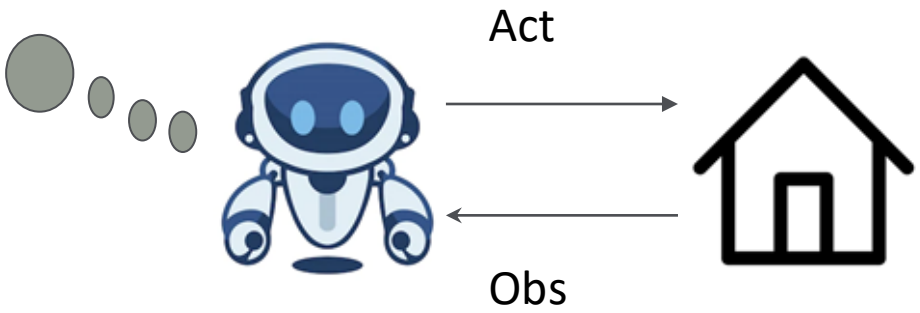
Date Understanding

Q: The concert was scheduled to be on 06/01/1943, but was delayed by one day to today. What is the date 10 days ago in MM/DD/YYYY?

A: One day after 06/01/1943 is 06/02/1943, so today is 06/02/1943. 10 days before today is 05/23/1943. So the answer is 05/23/1943.

Why is reasoning helpful for agents?

Obs t	You are cooking a dish and seeing salt is out…
Reasoning	"The dish should be savory, and since salt is out, I should find the soy sauce instead. It is in the cabinet to my right…"
Act t	Turn right
Obs t+1	You see a cabinet and a table…
Act t+1	Open cabinet



Reason to act

1. Generalization
2. Alignment

Observation to Action mapping could be hard

Reasoning without acting

Question: Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?

Standard

Answer: iPod

✗

Reason Only

Thought: Let's think step by step. Apple Remote was originally designed to interact with Apple TV. Apple TV can be controlled by iPhone, iPad, and iPod Touch. So the answer is iPhone, iPad, and iPod Touch.

Answer: iPhone, iPad, iPod Touch

✗

Reasoning with acting

Question: Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?

ReAct (Reason + Act)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.
Act 1: **Search**[Apple Remote]
Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple … originally designed to control the Front Row media center program ……(long text omitted)

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.
Act 2: **Search**[Front Row]
Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports','Front Row (software)', …]

Thought 3: Front Row is not found. I need to search Front Row (software) .
Act 3: **Search**[Front Row (software)]
Obs 3: Front Row is a discontinued media center software …(long text omitted)

Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.
Act 4: **Finish**[keyboard function keys]

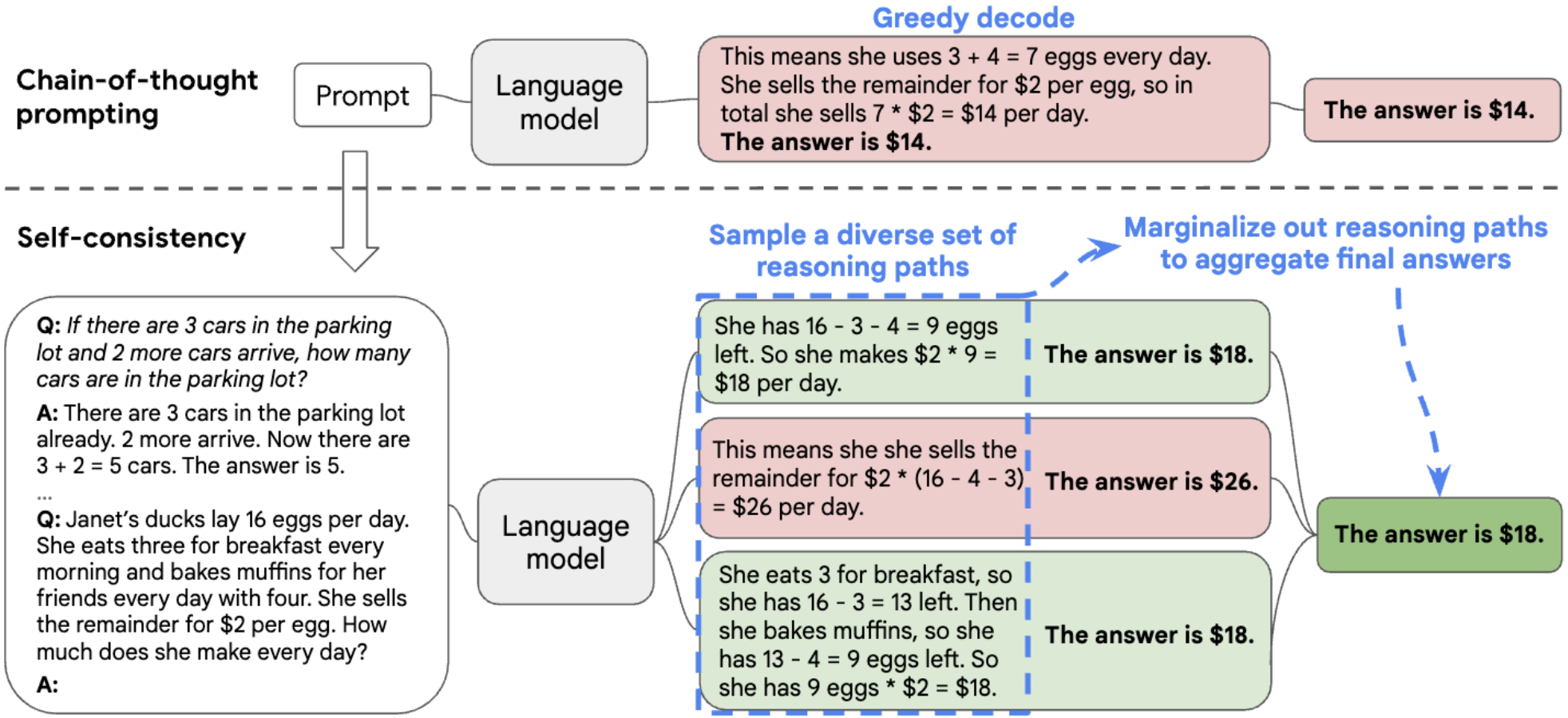
Comparing ReAct with COT, SC

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

- ReAct outperforms Act consistently
- Hallucination is a serious problem for CoT, and ReAct can help mitigate it to some extent

	Type	Definition	ReAct	CoT
Success	True positive	Correct reasoning trace and facts	94%	86%
	False positive	Hallucinated reasoning trace or facts	6%	14%
Failure	Reasoning error	Wrong reasoning trace (including failing to recover from repetitive steps)	47%	16%
	Search result error	Search return empty or does not contain useful information	23%	-
	Hallucination	Hallucinated reasoning trace or facts	0%	56%
	Label ambiguity	Right prediction but did not match the label precisely	29%	28%

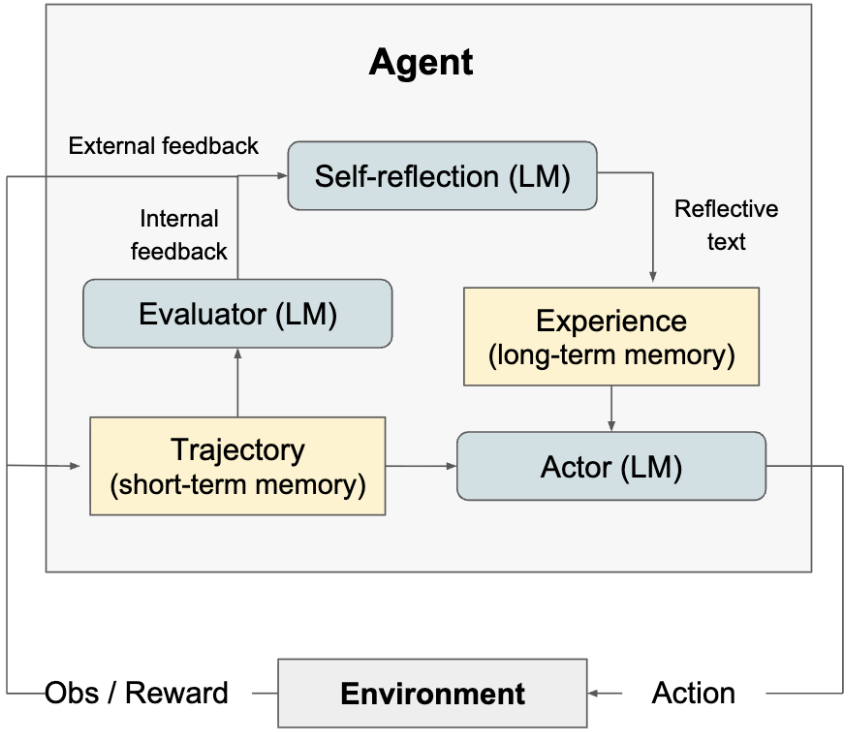
Self-Consistency [Wang et al., 2023]

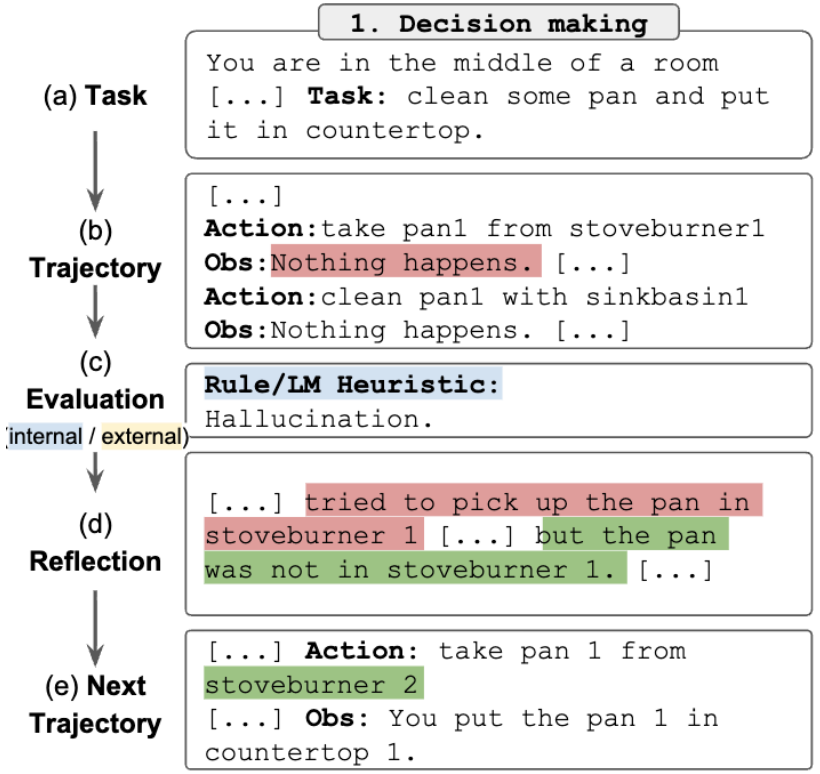


- Prompt a language model using chain-of-thought (CoT) prompting;
- Generate a diverse set of reasoning paths
- Aggregate by choosing the most consistent answer in the final answer set.

Reflextion: reinforce agents via natural language feedback

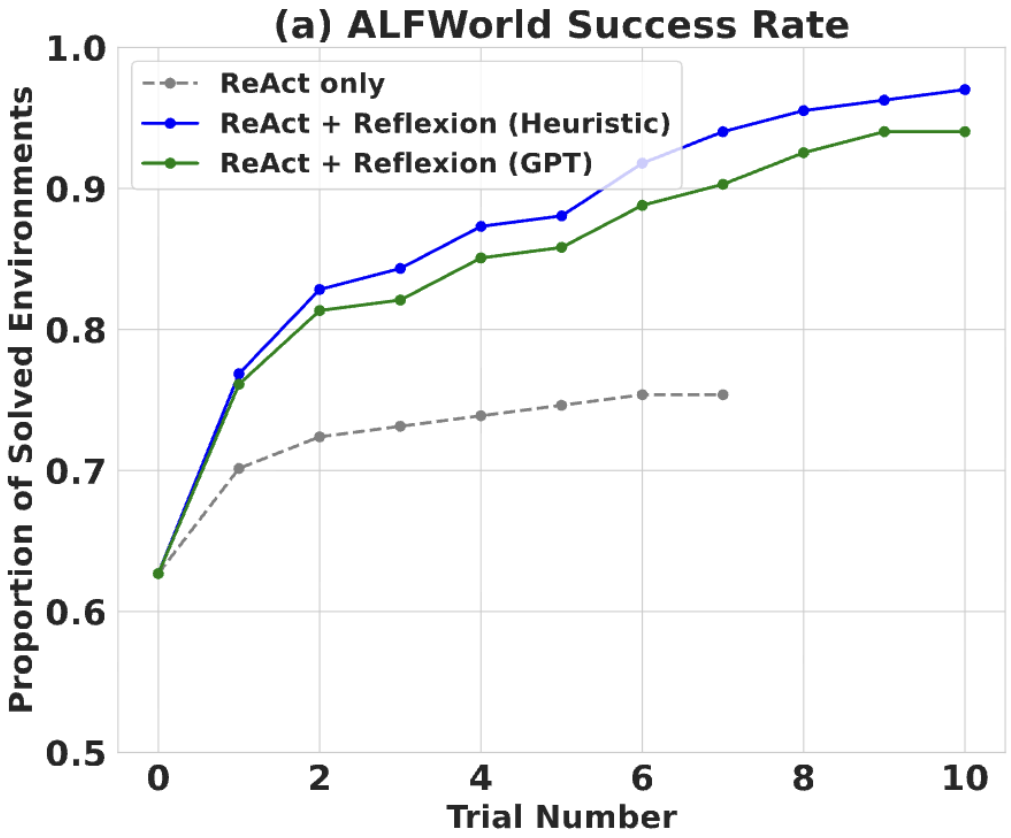
- Reflexion agents verbally reflect on task feedback signals, then maintain their own reflective text to induce better decision-making.

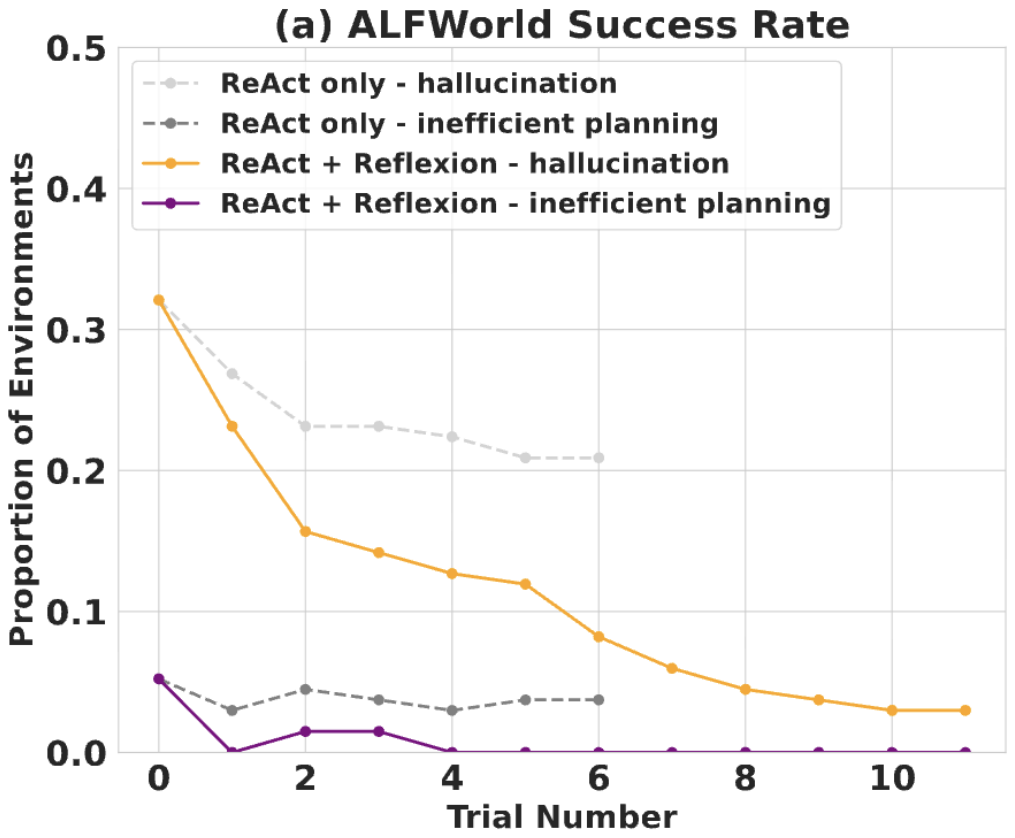




Reflextion: reinforce agents via natural language feedback

- Improve performance compared to ReAct
- Reduce error cases by using self-reflection to distill long, failed trajectories into relevant experiences that can are used as "self-hints" in the future





Multi-agent debate to improve reasoning [Du et al., 2023]

User Input: A treasure hunter found a buried treasure chest filled with gems. There were 175 diamonds, 35 fewer rubies than diamonds, and twice the number of emeralds than the rubies. How many of the gems were there in the chest?

Multi-agent debate to improve reasoning [Du et al., 2023]

Round 2

Agent 1: Given the information provided in other agents' solutions, we have two answers: One agent found that the number of rubies is 40 and the total number of gems is 225. Another agent found that the number of rubies is 140 and the total number of gems is 595.

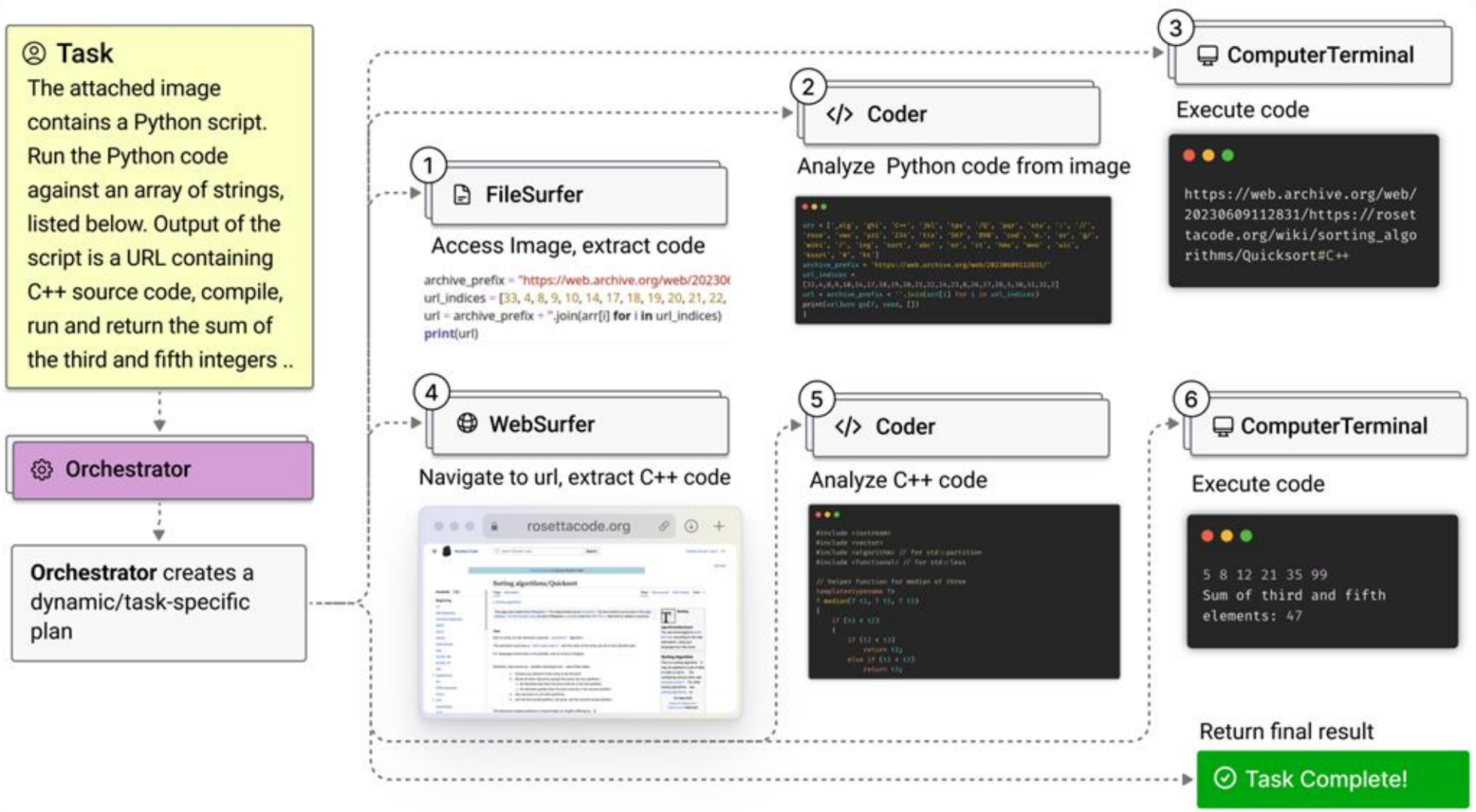
We know that the number of rubies is 35 fewer than the number of diamonds, which is 175. So the number of rubies should be 175 - 35 = 140, as the second agent found. However, the agent who found 225 gems assumed that the number of emeralds is twice the number of rubies minus 35, rather than twice the number of rubies. If we use this assumption, we get: Number of rubies = 140. Number of emeralds = 2 * 140 - 35 = 245. Total number of gems = 175 + 140 + 245 = 560. Our final answer is 560.

Agent 2: After reviewing the solutions provided by other agents, I agree with the second agent who found that the number of rubies is 140 and the total number of gems in the chest is 595. Therefore, my answer to the math problem is 595.

User Input: Using the solutions from other agents as additional information, can you give an updated response....

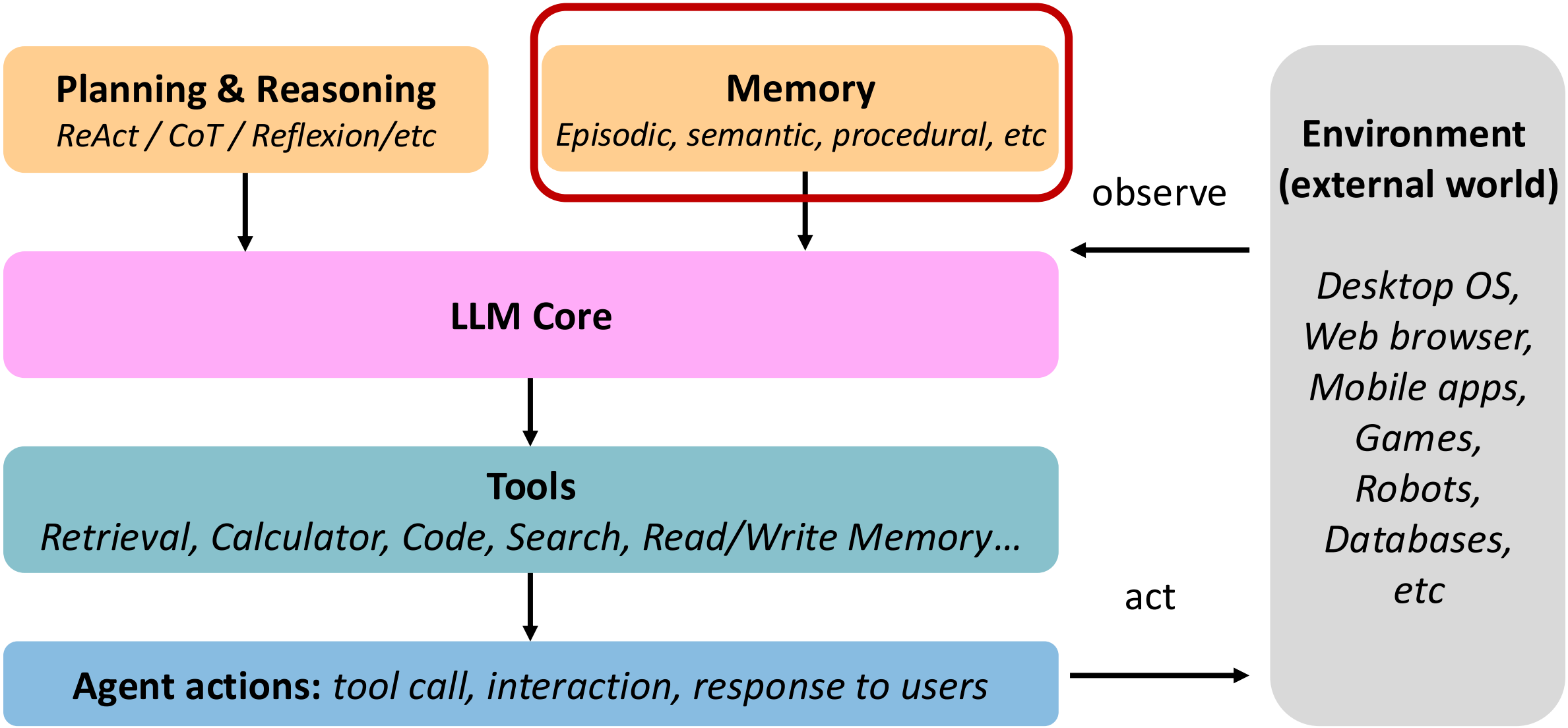
(Copy Agent 2's reply) (Copy Agent 1's reply)

Coordinate with Orchestrator



https://www.microsoft.com/en-us/research/publication/magentic-one-a-generalist-multi-agent-system-for-solving-complex-tasks/

4. Language agents: Memory

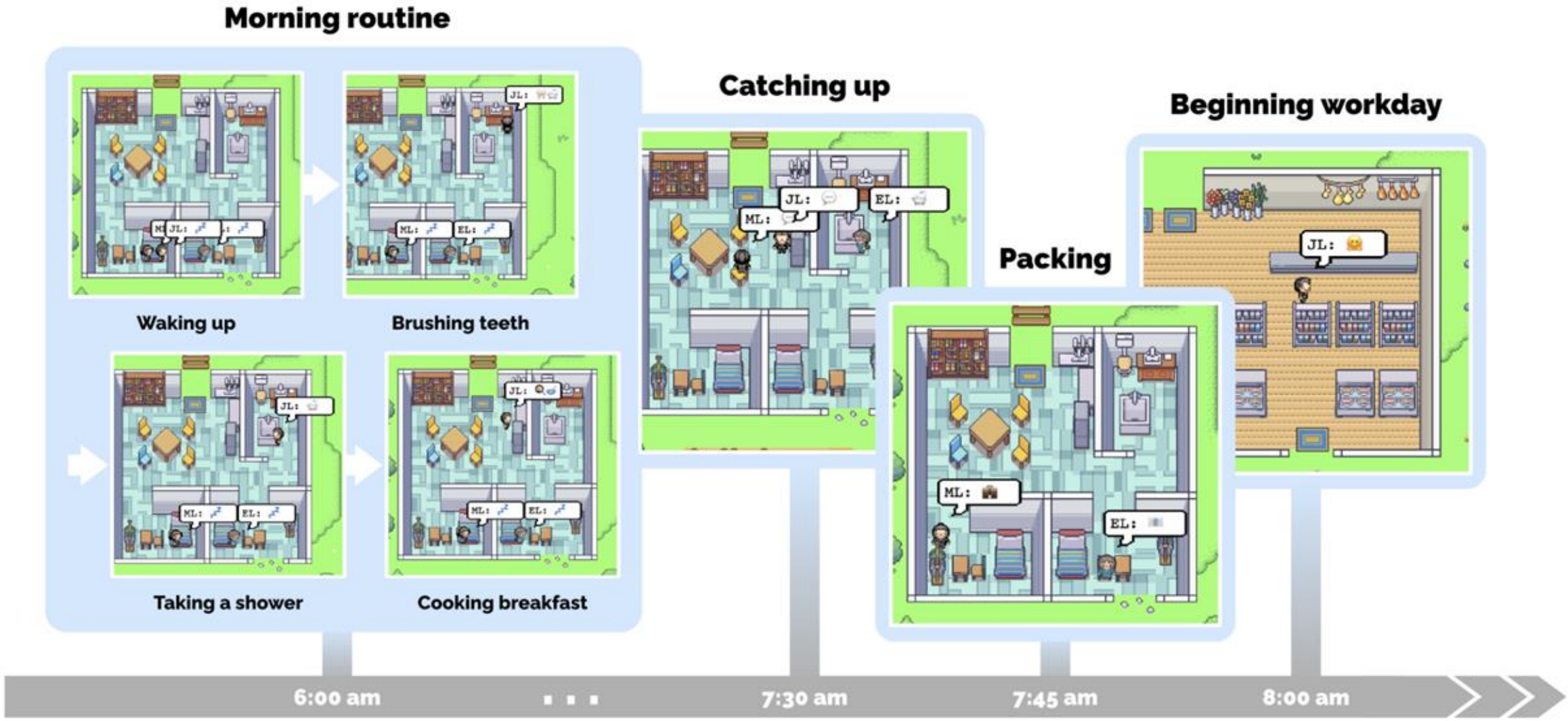


Different types of memory for language agents

Type by content	Definition	Examples
Episodic memory	Stores experience	Generative agents [Park et al., 2023]
Semantic memory	Stores knowledge	
Procedural memory	Stores skills	Voyager [Wang et al., 2023]

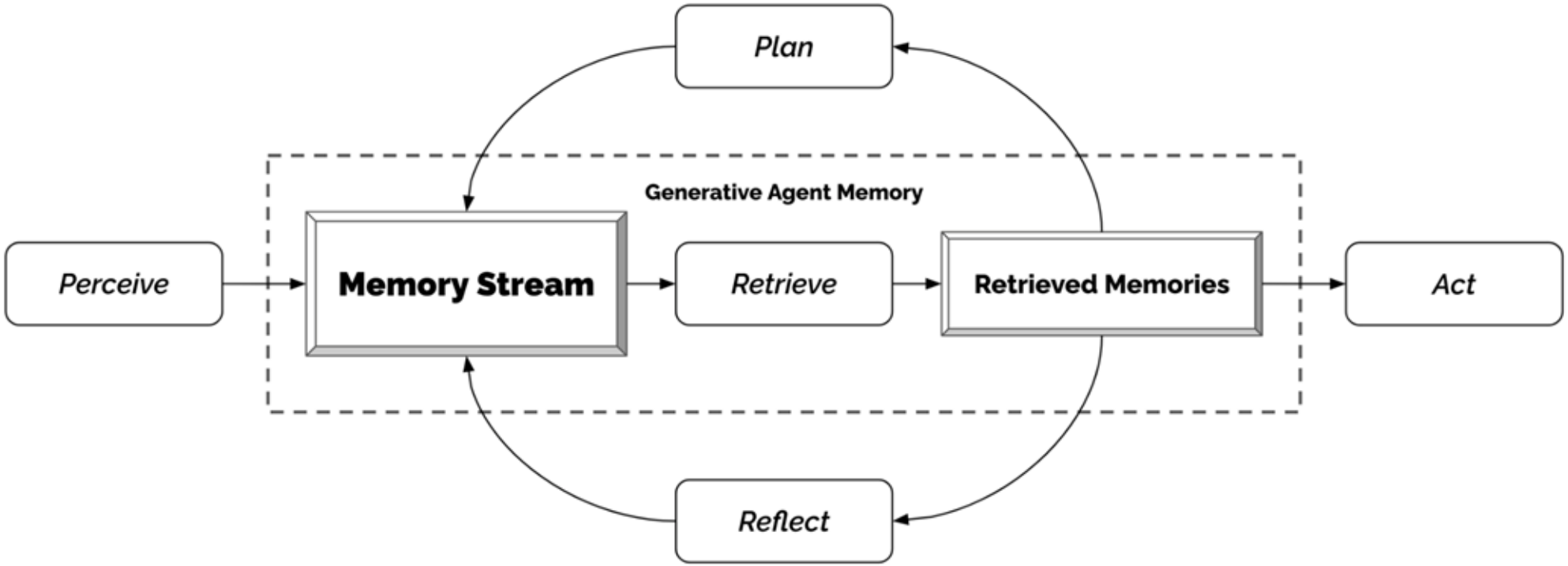
Note: here we only categorize based on memory content, which is inspired by human long-term memory systems

Generative agents



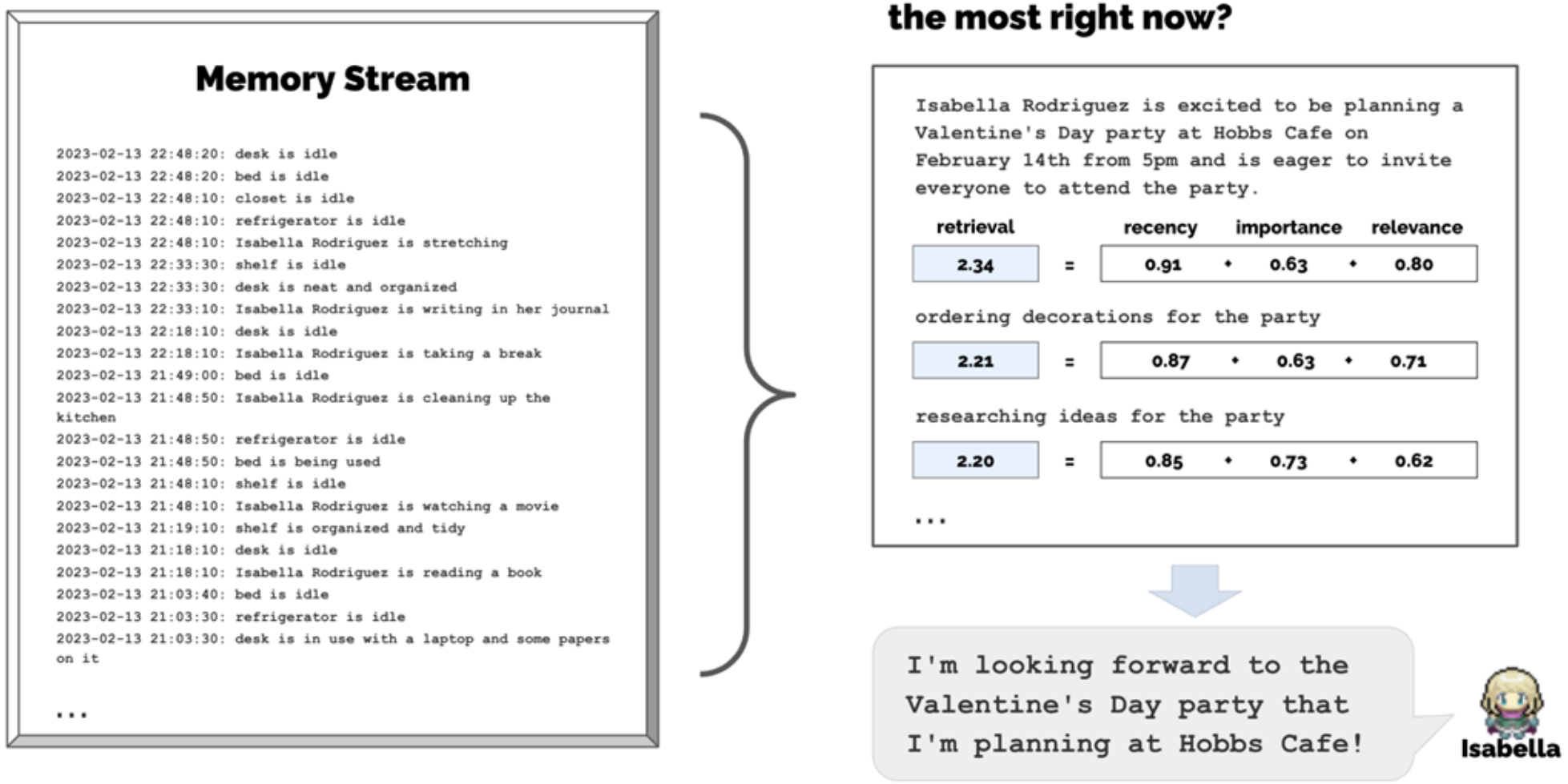
Generative agents: the need for memory

- Context window cannot possibly hold all the event streams
- Even if possible, might be hard to attend to relevant events, or digest over them



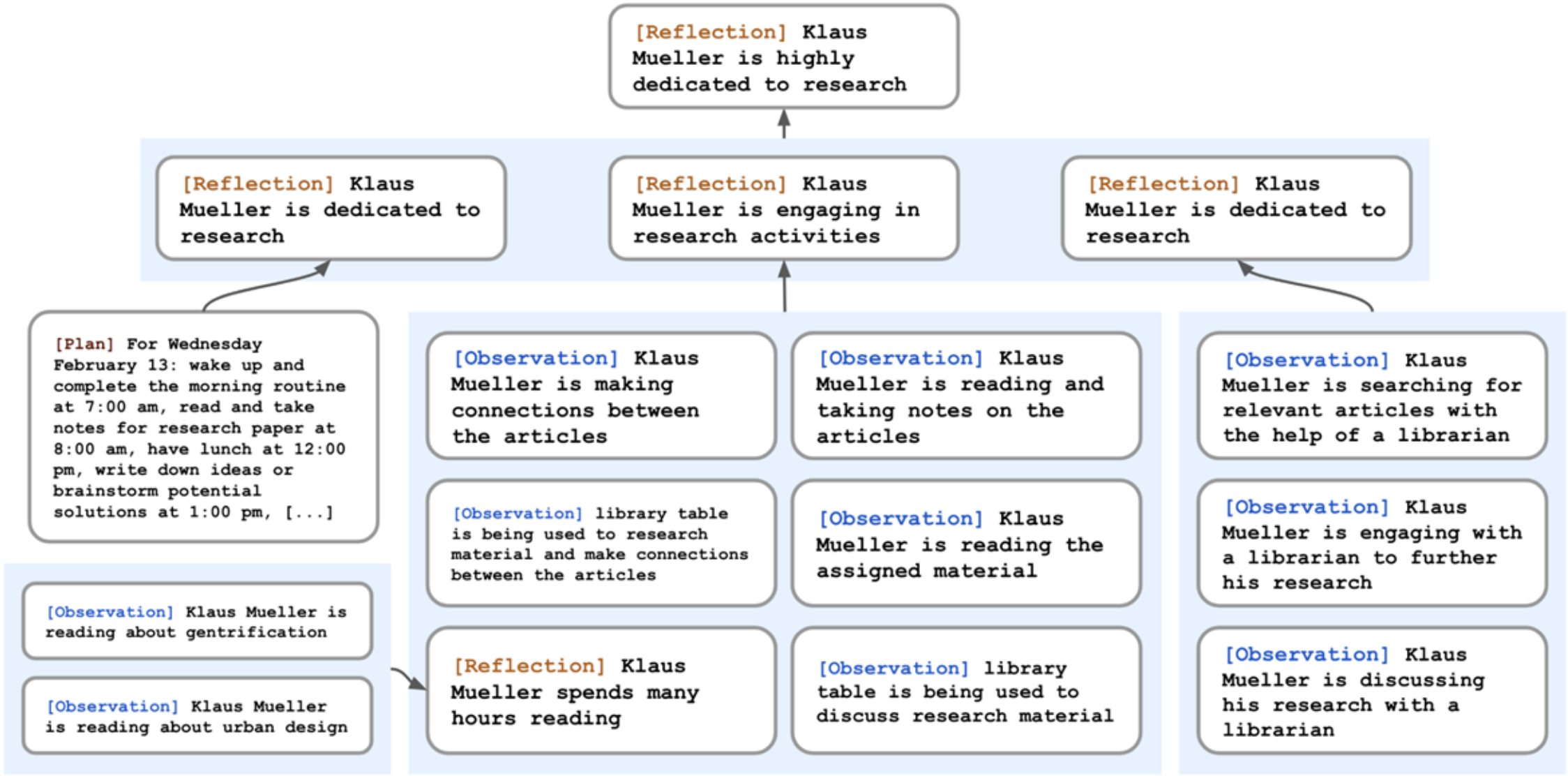
Episodic memory

- Write: append-only event streams
- Read: retrieval based on heuristic scores



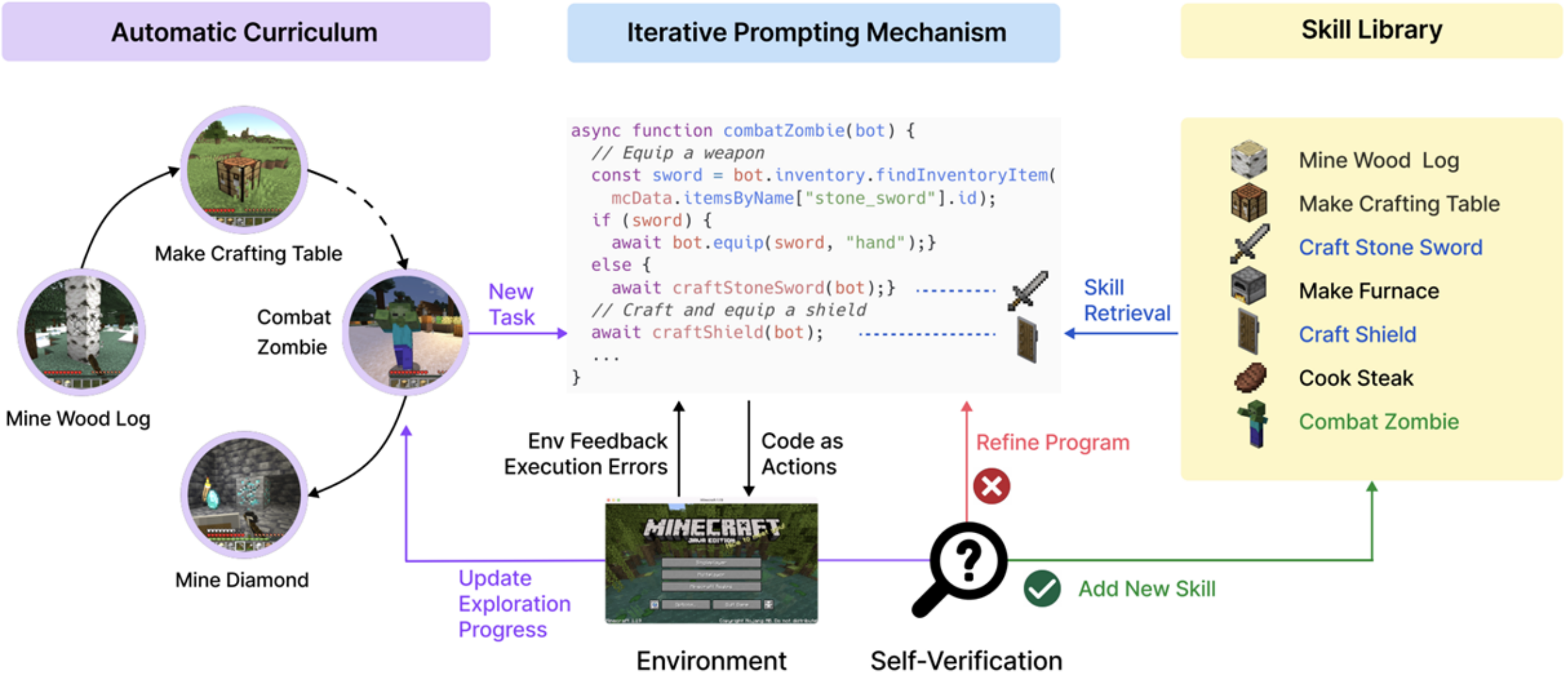
Semantic memory

- Write: LLM reasoning over events
- Read: retrieval



Procedural memory

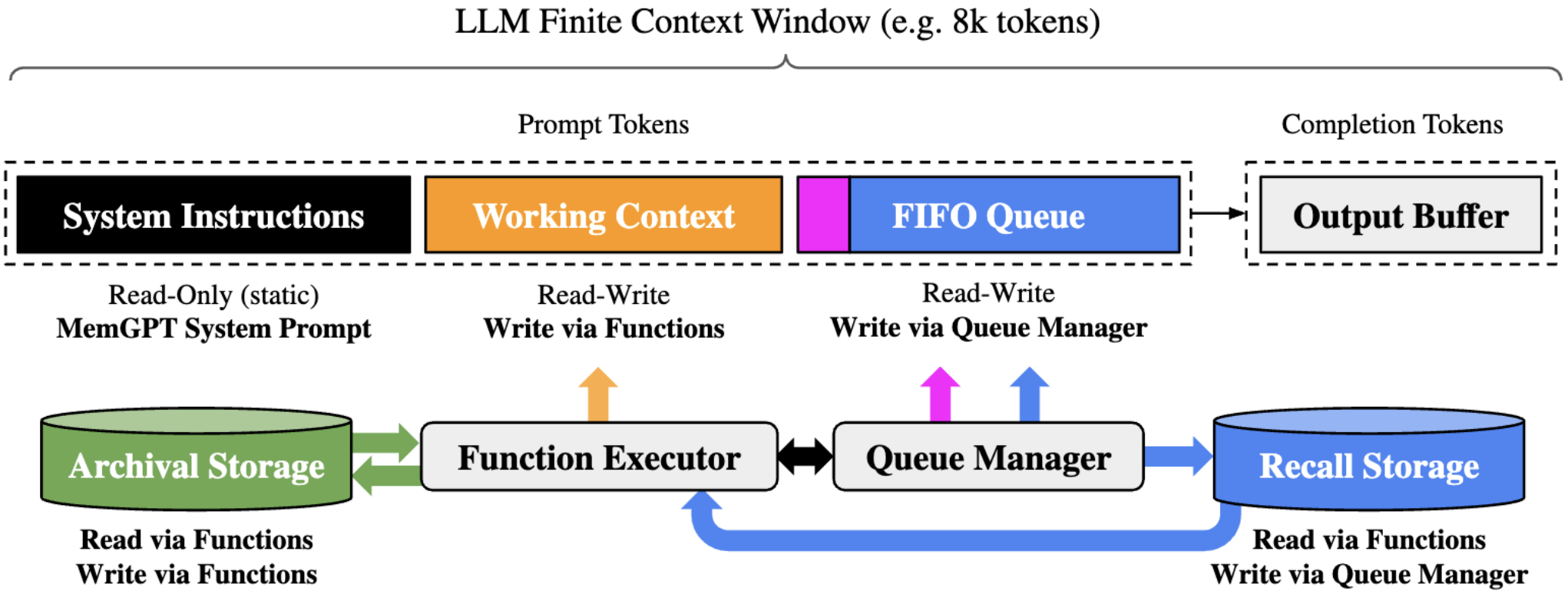
- Write: Code-based skills
- Read: Embedding retrieval



MemGPT [Packer+2024]

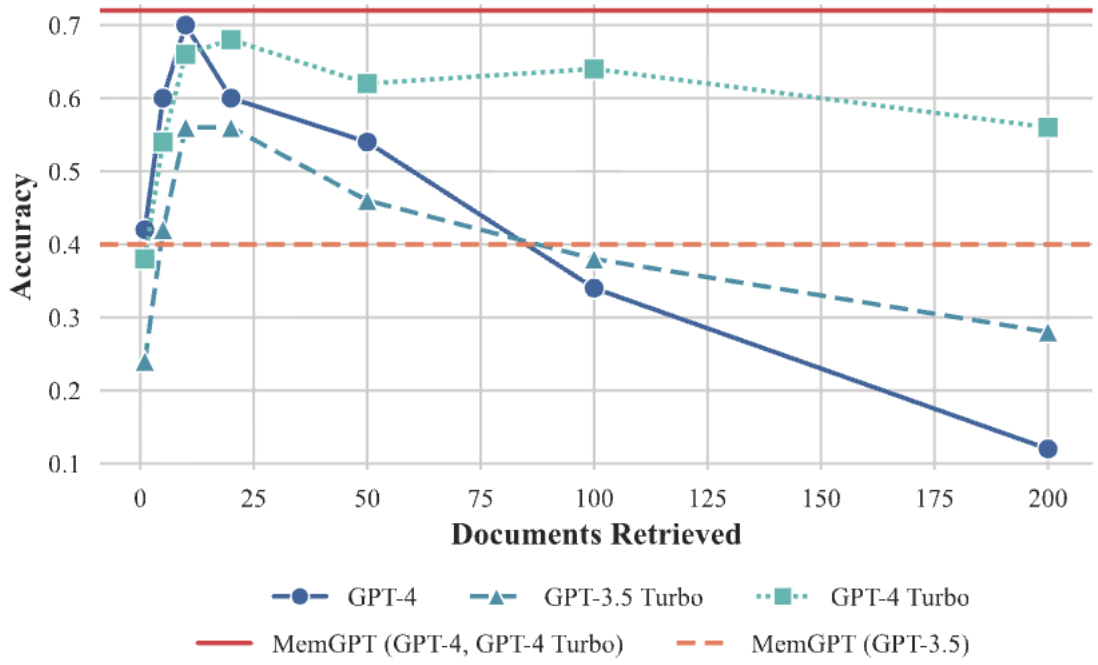
OS-inspired LLM system for virtual context management;

Using function calls, LLM agents can read and write to external data sources, modify their own context, and choose when to return responses to the user.



MemGPT [Packer+2024]

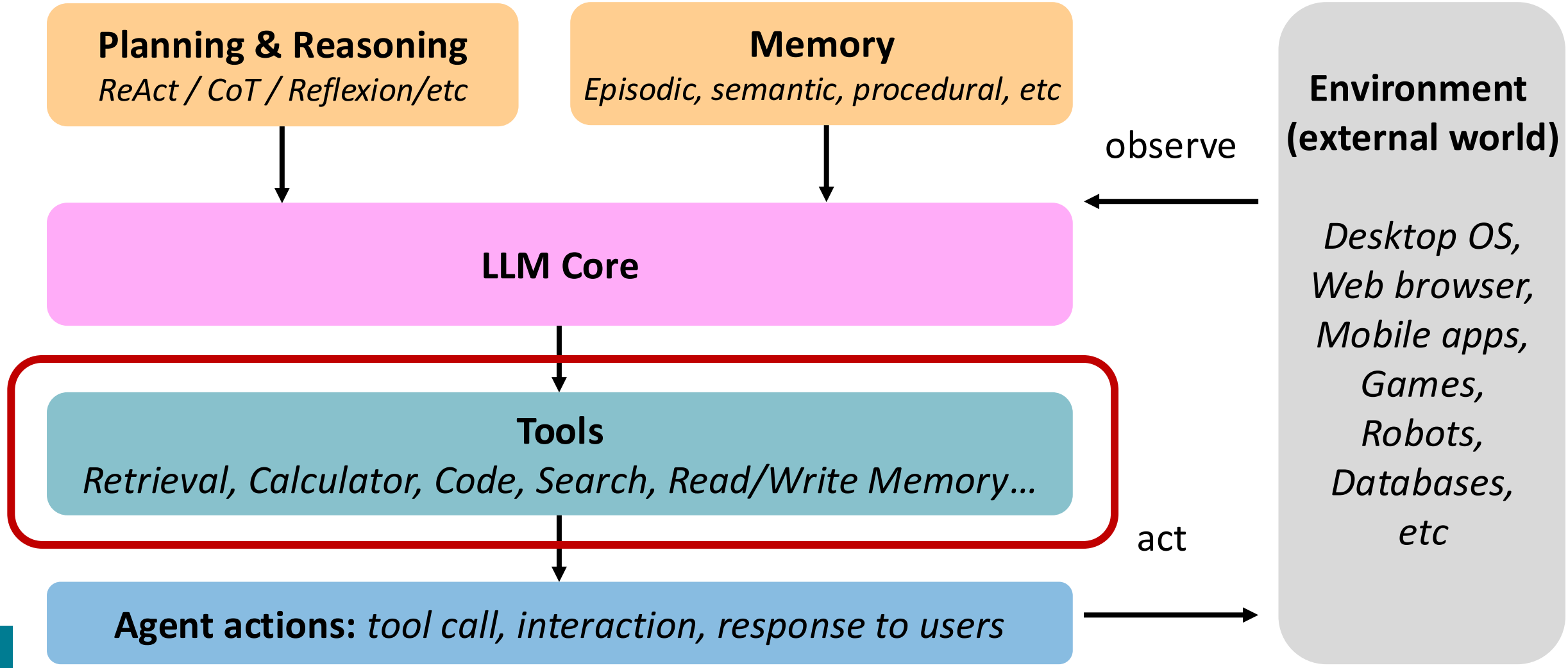
Model	Accuracy ⇑	ROUGE-L (R) ⇑
GPT-3.5 Turbo	38.7%	0.394
+ MemGPT	66.9%	0.629
GPT-4	32.1%	0.296
+ MemGPT	92.5%	0.814
GPT-4 Turbo	35.3%	0.359
+ **MemGPT**	**93.4%**	**0.827**



Deep memory retrieval (DMR) performance

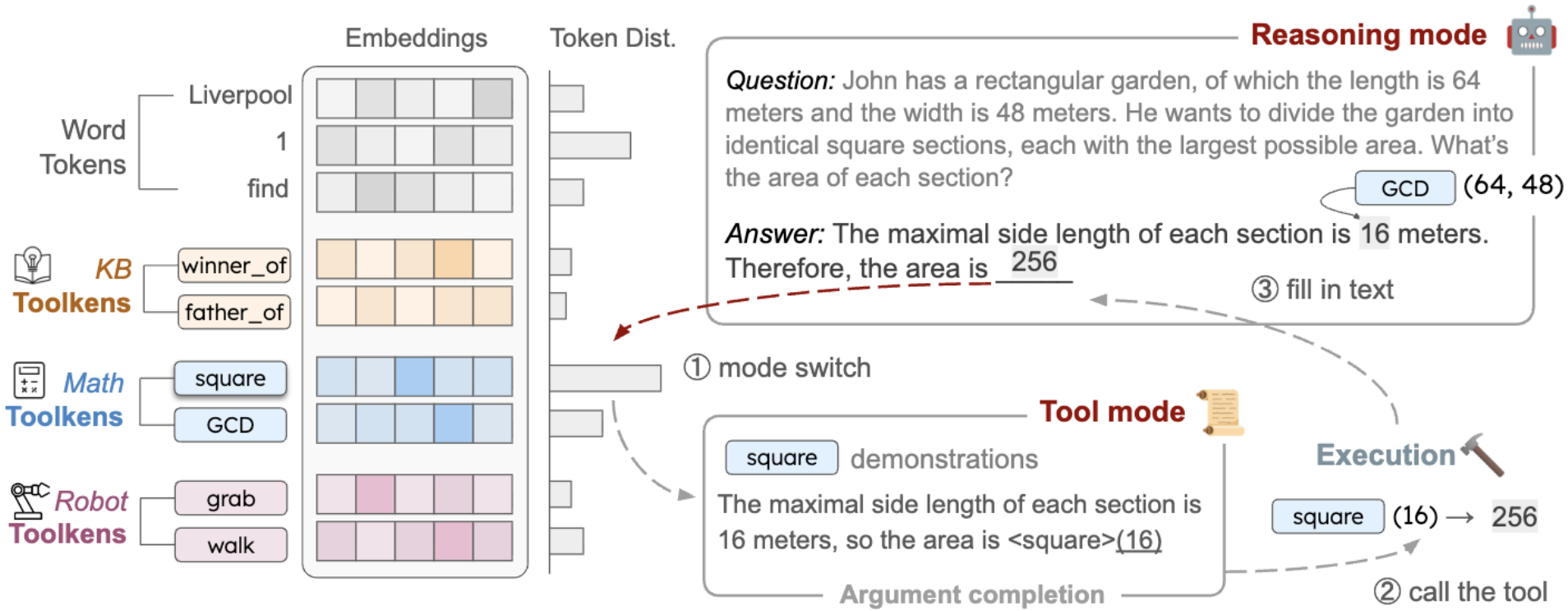
MemGPT's performance is unaffected by increased context length

5. Language agents: Tool use



Toolken GPT [Hao et al., 2024]

- ToolkenGPT represents each tool as a token ("toolken") and learns an embedding for it, enabling tool calls in the same way as generating a regular word token. Once a toolken is triggered, LM is prompted to complete arguments for the tool to execute.



Toolformer: language models can teach themselves to use tools

- Toolformer trains LMs to decide which APIs to call, when to call them, what arguments to pass, and how to best incorporate the results.
- Use self-supervised learning with only a handful of demonstrations for each API.
- Tools: calculator, Q&A system, search engine, translation system, calendar.

The New England Journal of Medicine is a registered trademark of **[QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society]** the MMS.

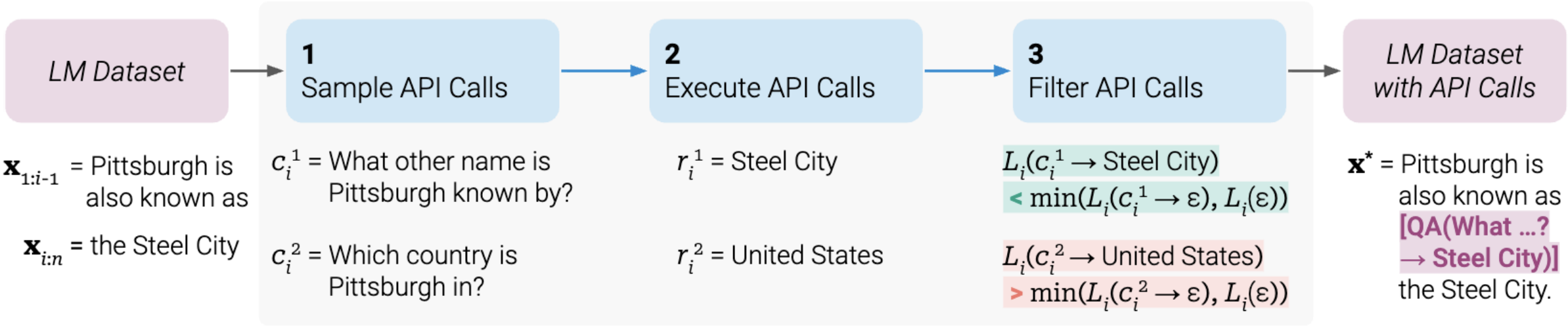
Out of 1400 participants, 400 (or **[Calculator(400 / 1400) → 0.29]** 29%) passed the test.

The name derives from "la tortuga", the Spanish word for **[MT("tortuga") → turtle]** turtle.

The Brown Act is California's law **[WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.]** that requires legislative bodies, like city councils, to hold their meetings open to the public.

Toolformer: language models can teach themselves to use tools

- For an input $\boldsymbol{x}$, sample a position i and corresponding API call candidates $c_i^1, c_i^2, \ldots c_i^k$.
- We then execute these API calls and filter out all calls which do not reduce the loss L_i over the next tokens.
- All remaining API calls are interleaved with the original text, resulting in a new text $\boldsymbol{x}^*$



Toolformer: language models can teach themselves to use tools

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

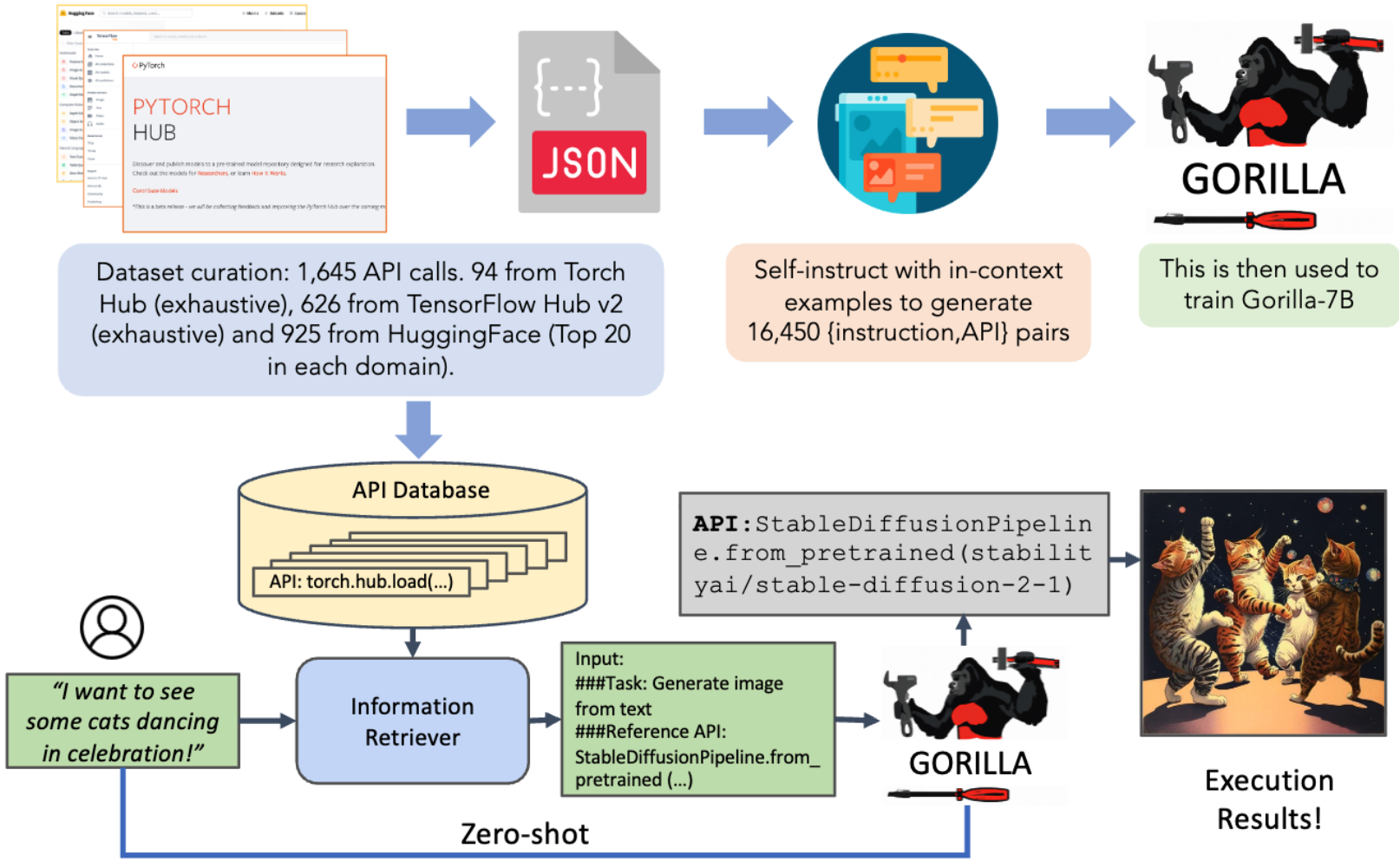
Output:

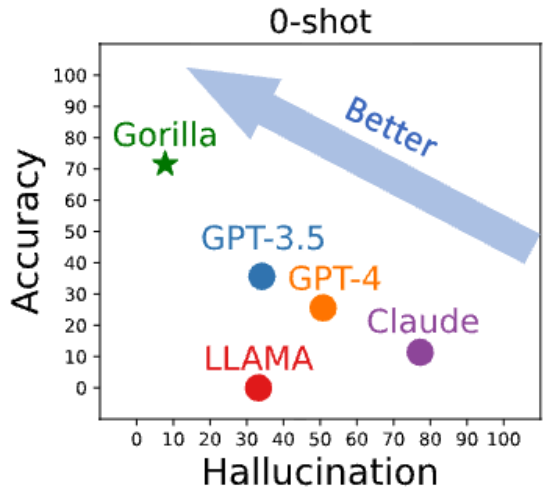
Model	SQuAD	Google-RE	T-REx
GPT-J	17.8	4.9	31.9
GPT-J + CC	19.2	5.6	33.2
Toolformer (disabled)	22.1	6.3	34.9
Toolformer	**33.8**	**11.5**	**53.5**
OPT (66B)	21.6	2.9	30.1
GPT-3 (175B)	26.8	7.0	39.8

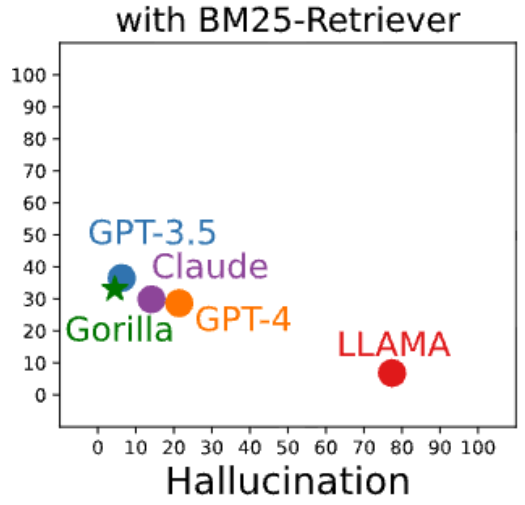
Model	WebQS	NQ	TriviaQA
GPT-J	18.5	12.8	43.9
GPT-J + CC	18.4	12.2	45.6
Toolformer (disabled)	18.9	12.6	46.7
Toolformer	**26.3**	**17.7**	**48.8**
OPT (66B)	18.6	11.4	45.7
GPT-3 (175B)	29.0	22.6	65.9

Gorilla: LLMs connected with massive APIs (Patil et al., 2023)

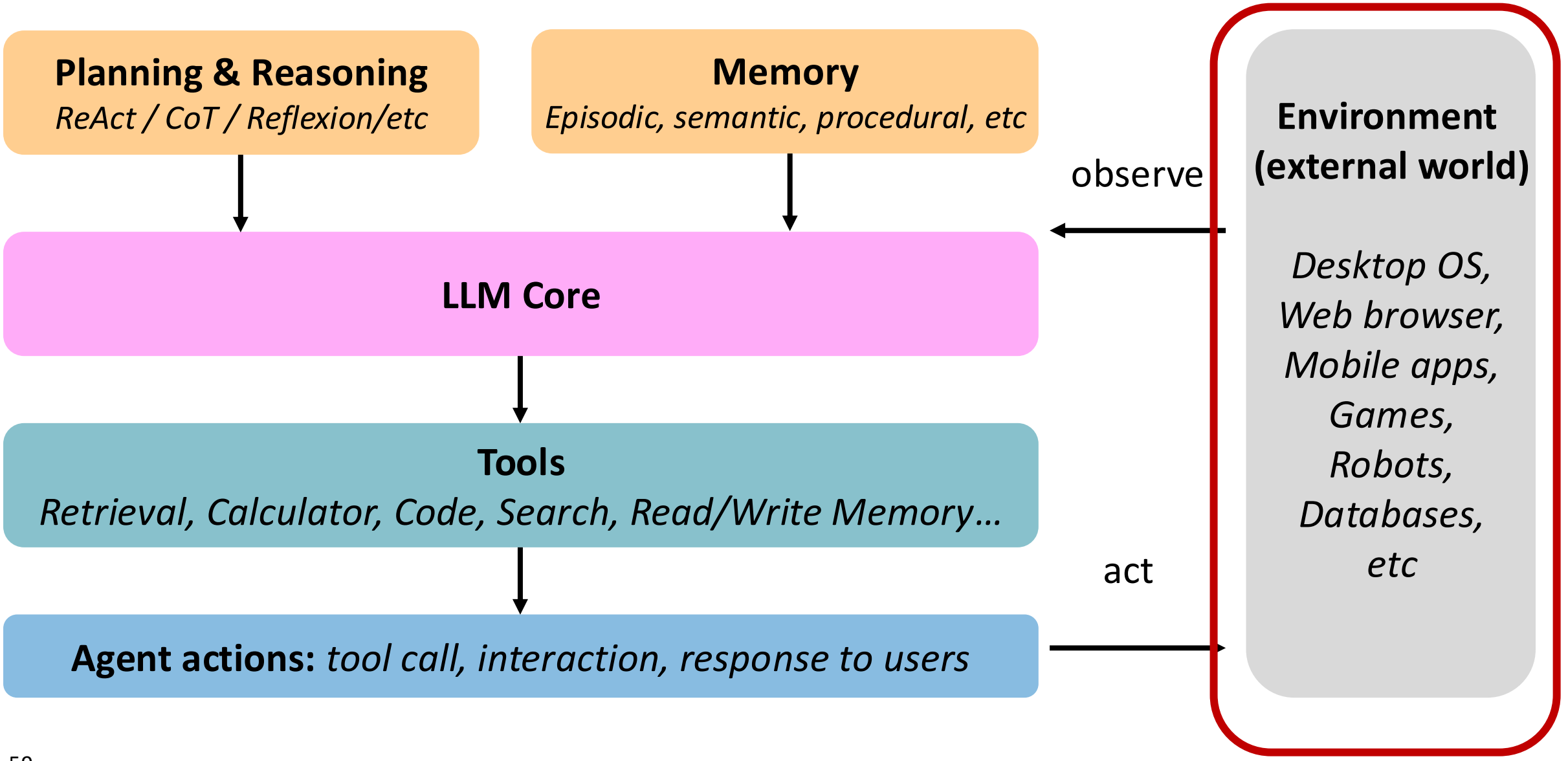
- Use self-instruct to generate {instruction, API} pairs and fine-tune LLaMa on it







6. Different agent applications, data and evaluation



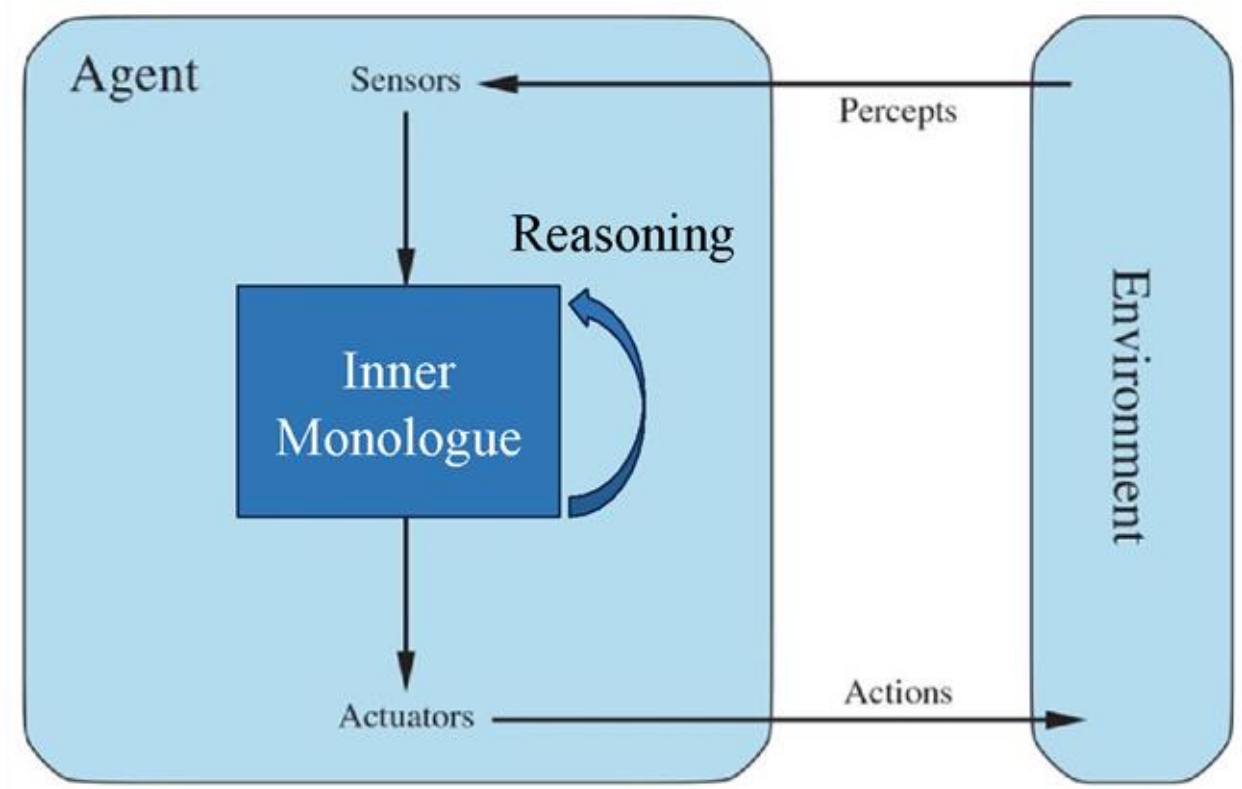
Agent applications based on environments and domains

Digital world

- Coding agents
- Gaming agents
- Mobile agents
- Web/app agents
- Computer agents

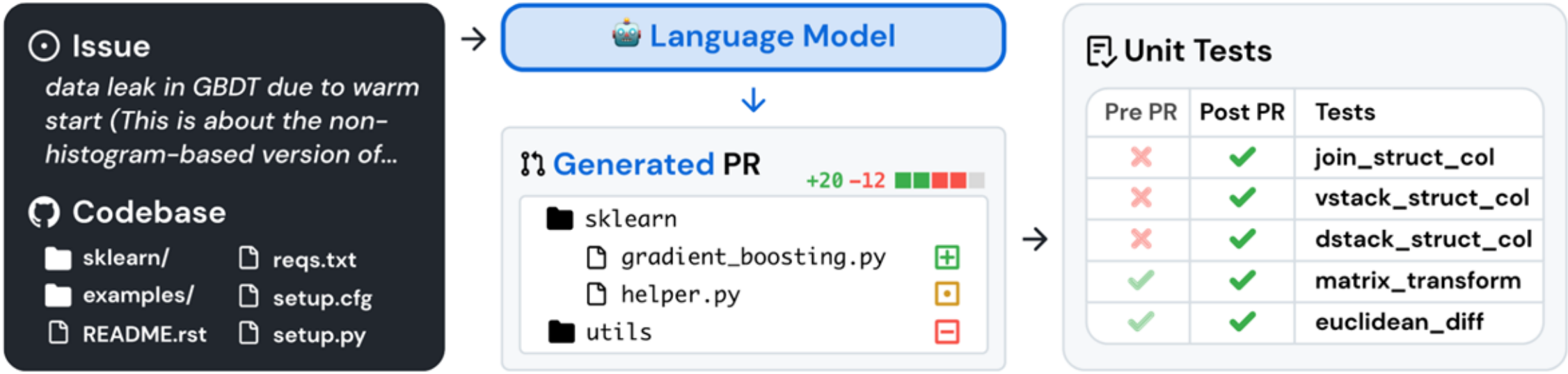
Physical world

- Robotics



Coding agents

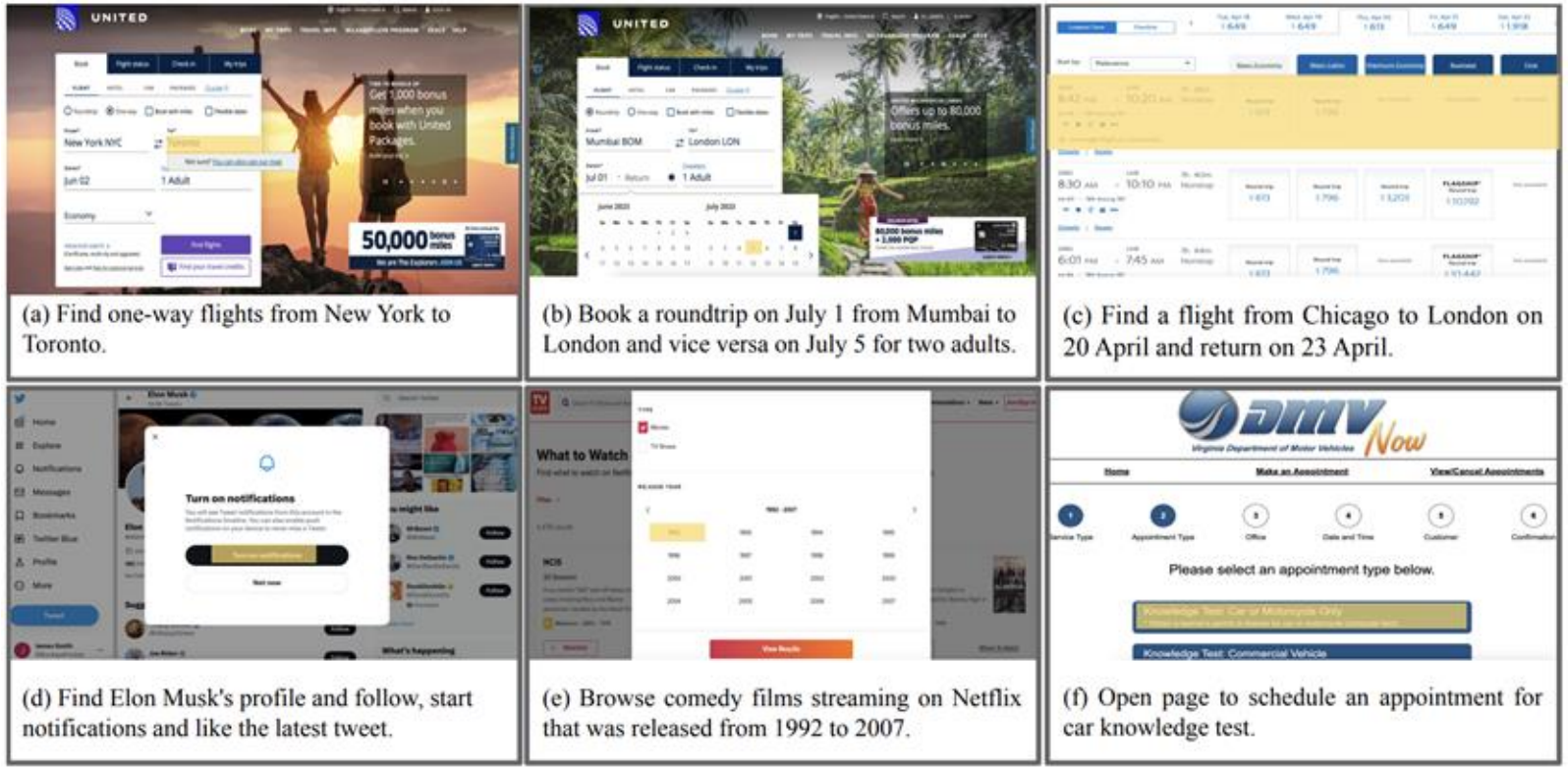
- Environment: project code repos, filesystems, IDEs…
- Observation space: code files, exe outputs, docs, errors, commit history…
- Action space: code edits, file search/view, test updates…



SWE-bench (Jimenez & Yang, 2023)

Web/app agents

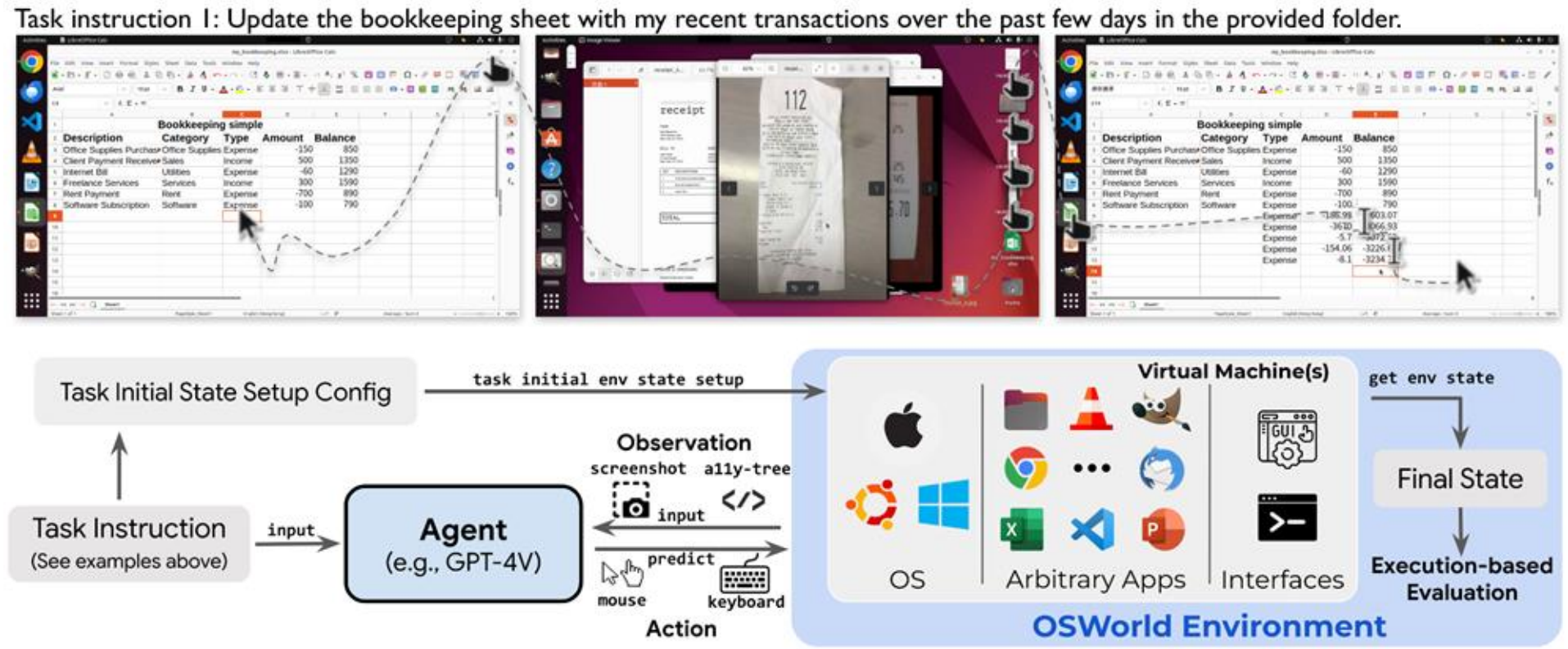
- Environment: web browsers/apps
- Observation space: screenshots, DOM trees, HTML, historical actions…
- Action space: browser/app controls (e.g., click, type, scroll, drag, hover…)



Mind2Web: Towards a Generalist Agent for the Web, (Deng et al., 2023)
WebArena: A Realistic Web Environment for Building Autonomous Agents, (Zhou et al., 2023)

Computer use agents

- Environment: desktop operating systems
- Observation space: desktop screenshots, a11y trees, historical actions…
- Action space: keyboard/mouse controls (e.g., click, type, drag, shortcuts)

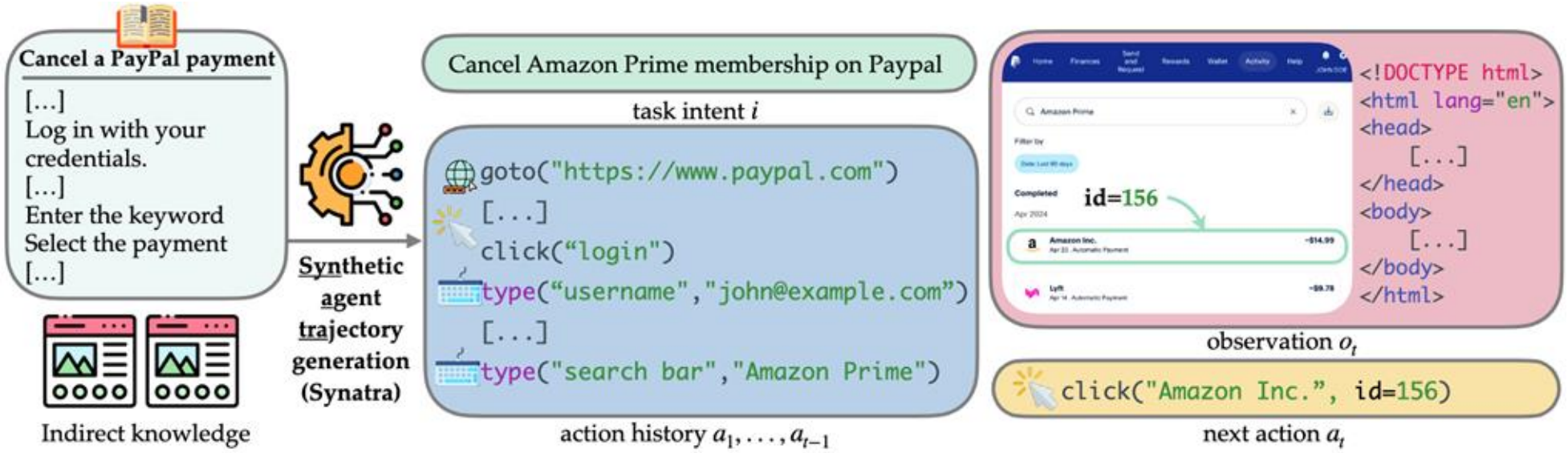


OpenAI Universe, (OpenAI, 2016)
OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments, (Xie et al., 2024)

Agent data --- scaling agent data

- Human demonstrations
  - Expensive and complex infrastructure setup
  - Expert time & cost
  - Task coverage

- Synthesis/simulation
  - Converting online tutorials into direct training demonstrations [Synatra]

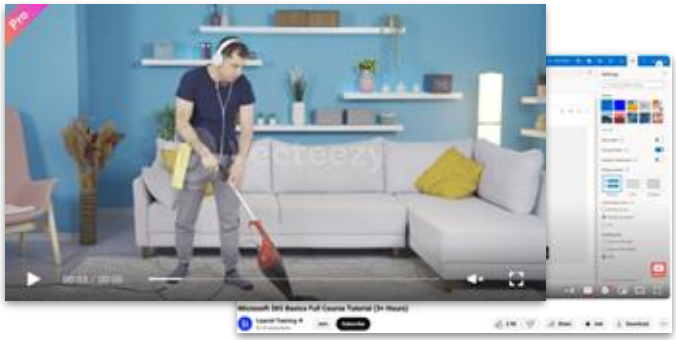


Agent data --- scaling agent data

- Human demonstrations
  - Expensive and complex infra
  - Expert time & cost
  - Task coverage

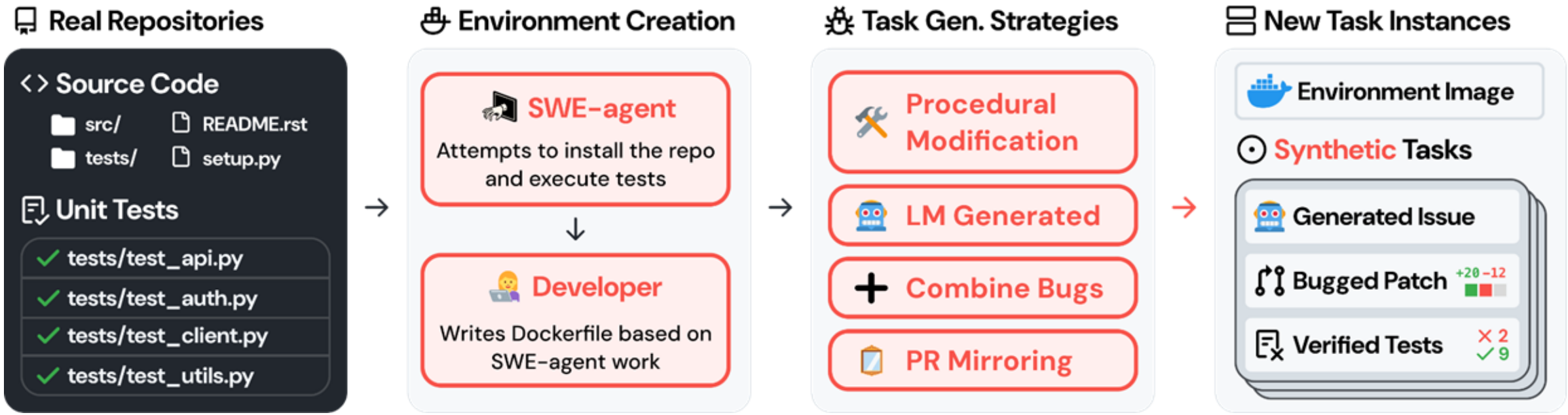
- Synthesis/simulation
  - Converting online tutorials into direct training demonstrations [Synatra]

- Internet-scale data
  - Numerous videos/data exist online showing how humans perform (agent) tasks, but without grounded trajectories



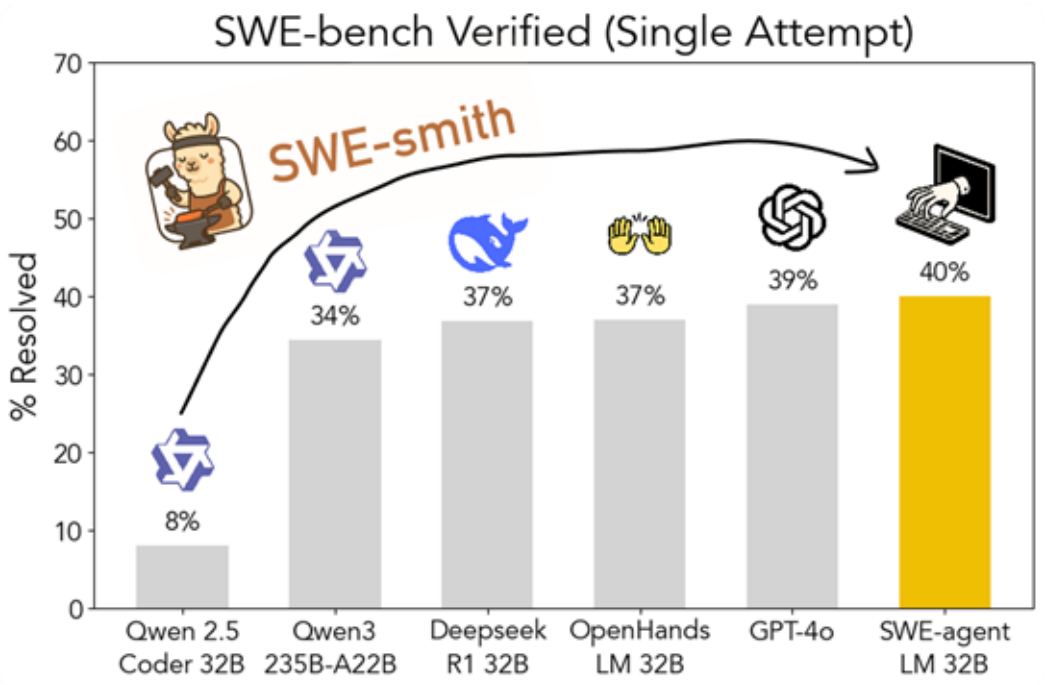
Case study of SWE-smith: scaling data for coding agents

A toolkit for (1) creating execution environments, and (2) synthesizing 100s to 1000s of task instances for any (Python) GitHub repository [Yang et al., 2025]



Training details and results

- **Training Technique**: Rejection Sampling Fine Tuning
- **Models**: Claude 3.5/3.7, GPT 4o (Experts); Qwen 2.5 Coder Instruct (Student)
- **Agent System**: SWE-agent



Model	System	Train Size	Lite	Verified
Closed Weight Models				
GPT-4o (OpenAI, 2024a)	Agentless	-	32.0	38.8
	OpenHands	-	22.0	-
	SWE-agent	-	18.3	23.0
Claude 3.5 Sonnet (Anthropic, 2024)	Agentless	-	40.7	50.8
	AutoCodeRover	-	-	46.2
	OpenHands	-	41.7	53.0
	SWE-agent	-	23.0	33.6
Claude 3.7 Sonnet (Anthropic, 2025)	SWE-agent	-	**48.0**	**58.2**
Llama3-SWE-RL-70B (Wei et al., 2025)	Agentless	11M	-	41.0
Open Weight Models				
Lingma-SWE-GPT-72B (Ma et al., 2024)	SWE-SynInfer	-	-	28.8
Qwen3-235B-A22B (Qwen et al., 2025)	OpenHands	-	-	34.4
R2E-Gym-32B (Jain et al., 2025)	OpenHands	3.3k	-	34.4
SWE-fixer-72B (Xie et al., 2025a)	SWE-Fixer	110k	24.7	32.8
SWE-gym-32B (Pan et al., 2024)	OpenHands	491	15.3	20.6
SWE-agent-LM-7B	SWE-agent	2k	11.7	15.2
SWE-agent-LM-32B	SWE-agent	5k	**30.7**	**40.2**

Agent evaluation

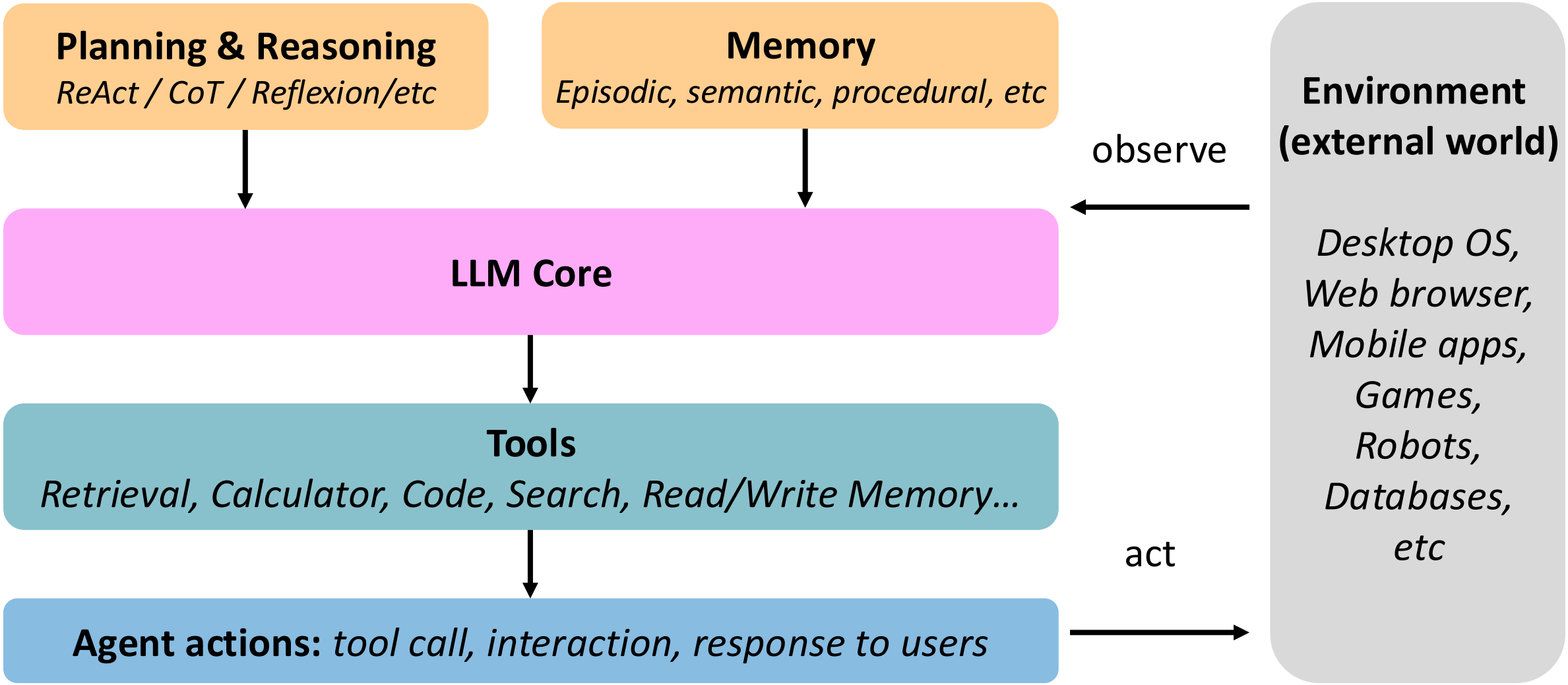
Challenges in agent evaluation

- Real-world environmental setup complexity
- Task coverage
- Open-ended success criteria
  - Multiple valid solution paths
  - Cannot script evaluation metrics, need for human judgment
- Evaluation beyond task success

Agent evaluation:

- via benchmarks; via LLMs/VLMs; via crowdsourcing
- See lecture on “Benchmarking and Evaluation”

Language agents: from words to action



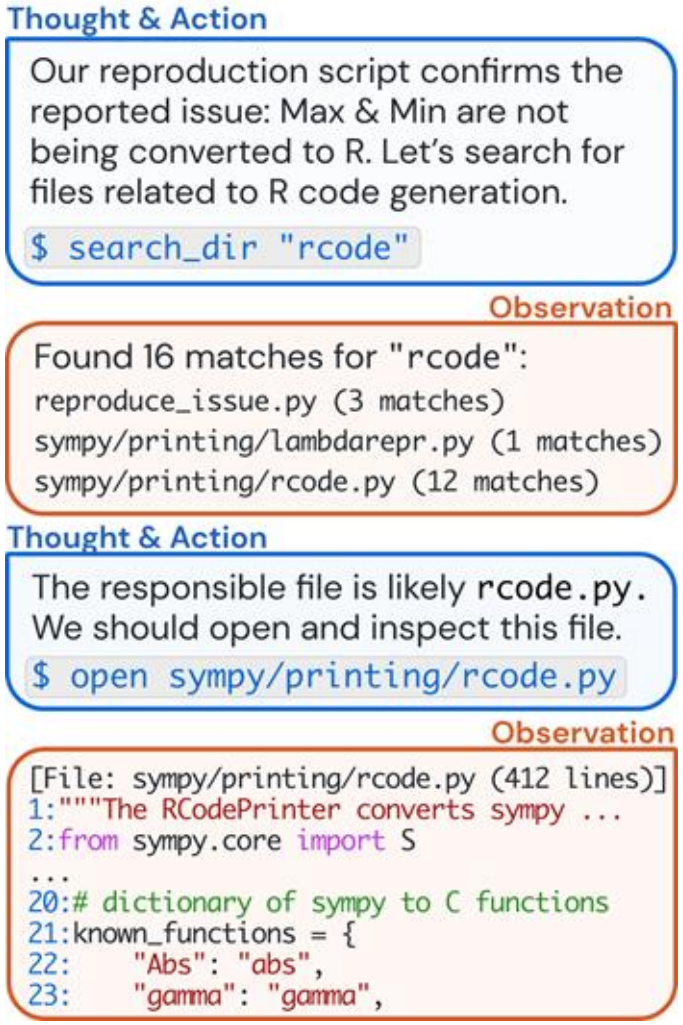
Bonus: building a coding agent (step by step)

John Yang

At its simplest, ***an AI coding agent is just a big loop***

Given a task, e.g. “create a web app for me”, we kickstart the following loop:

1. The LM proposes an action
2. You execute the action in a code environment (e.g., bash terminal, docker container, python interpreter)
3. Tell the LM the output
4. And then repeat steps 2-4!



Bonus: building a coding agent (step by step)

At its simplest, *An AI coding agent is just a big loop*. Some pseudocode:

```
messages = [{"role": "user", "content": "Help me fix the ValueError in main.py"}]
while True:
    lm_output = query_lm(messages)
    print("LM output", output)
    messages.append({"role": "assistant", "content": lm_output})  # remember what the LM said
    action = parse_action(lm_output)  # separate the action from output
    print("Action", action)
    if action == "exit":
        break
    output = execute_action(action)
    print("Output", output)
    messages.append({"role": "user", "content": output})  # send command output back
```

Let's implement 3 things: **query_lm**; **parse_action**; **execute_action**

Bonus: building a coding agent (step by step)

query_lm

Input: messages
Output: LM's response

```python
from openai import OpenAI

client = OpenAI(
    api_key="your-api-key-here"
)  # or set OPENAI_API_KEY env var

def query_lm(messages):
    response = client.responses.create(
        model="gpt-5.1",
        input=messages
    )
    return response.output_text
```

parse_action

Input: LM's response
Output: bash action

```python
import re

def parse_action(lm_output: str) -> str:
    """Take LM output, return action"""
    matches = re.findall(
        r"<bash_action>(.*?)</bash_action>",
        lm_output,
        re.DOTALL
    )
    return matches[0].strip() if matches else ""
```

execute_action

Input: base action
Output: std. output

```python
import subprocess
import os

def execute_action(command: str) -> str:
    """Execute action, return output"""
    result = subprocess.run(
        command,
        shell=True,
        text=True,
        env=os.environ,
        encoding="utf-8",
        errors="replace",
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
        timeout=30,
    )
    return result.stdout
```

Full tutorial here: https://minimal-agent.com/