

How to AI an AI for Almost Anything

Lecture 4 - Harness Evolution

Paul Liang

Assistant Professor

MIT Media Lab & MIT EECS



<https://pliang279.github.io>

ppliang@mit.edu

 [@pliang279](https://twitter.com/pliang279)



Announcements

- Homework 1 released, due Oct 13 11:59pm
- Reading 2 will go out later this week with submissions + presentation Oct 14
- Work on projects, next checkpoint is midterm report + presentation Oct 28
 - Processed datasets, ran baselines, done error analysis, better proposed idea
 - Bottom-up research. At the same time, continue top-down ideation from readings and literature review.

Today's Lecture

- Harness components, including skills, tools, memory, agent loops
- Memory and memory evolution
- Tools, skills, and their evolution
- Guest speaker: Recursive Language Models and harness evolution

Recap: Context Evolution

Context: Everything the model reads before generating the next token

types

developer

system prompt

instructions

tool definitions

examples

memory

user

user message

environment

files

search results

tool output

the model

its own thoughts and actions

uses

Inform

new information

Steer

instructions, role, format

Demonstrate

examples

Compute

thinking step by step

Correct

feedback from the last attempt

LLMs can be used to automatically optimize context, through search/optimization over prompts/text/modules.

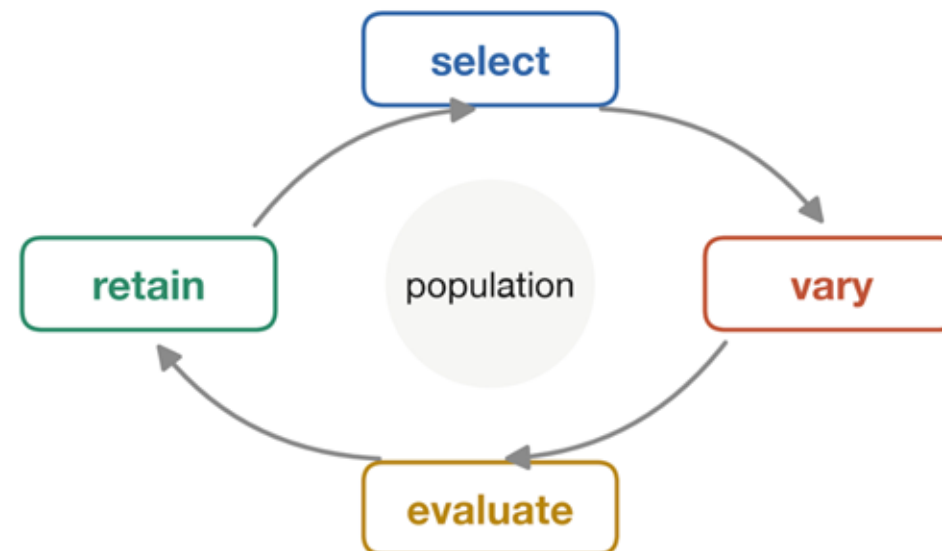
Recap: Context Evolution

LLMs can also evolve its own context (i.e., solution space) to solve open-ended problems (e.g., NP-hard optimization problems, coding, scientific discovery).

$$p^* = \arg \max_p \text{score}(p)$$

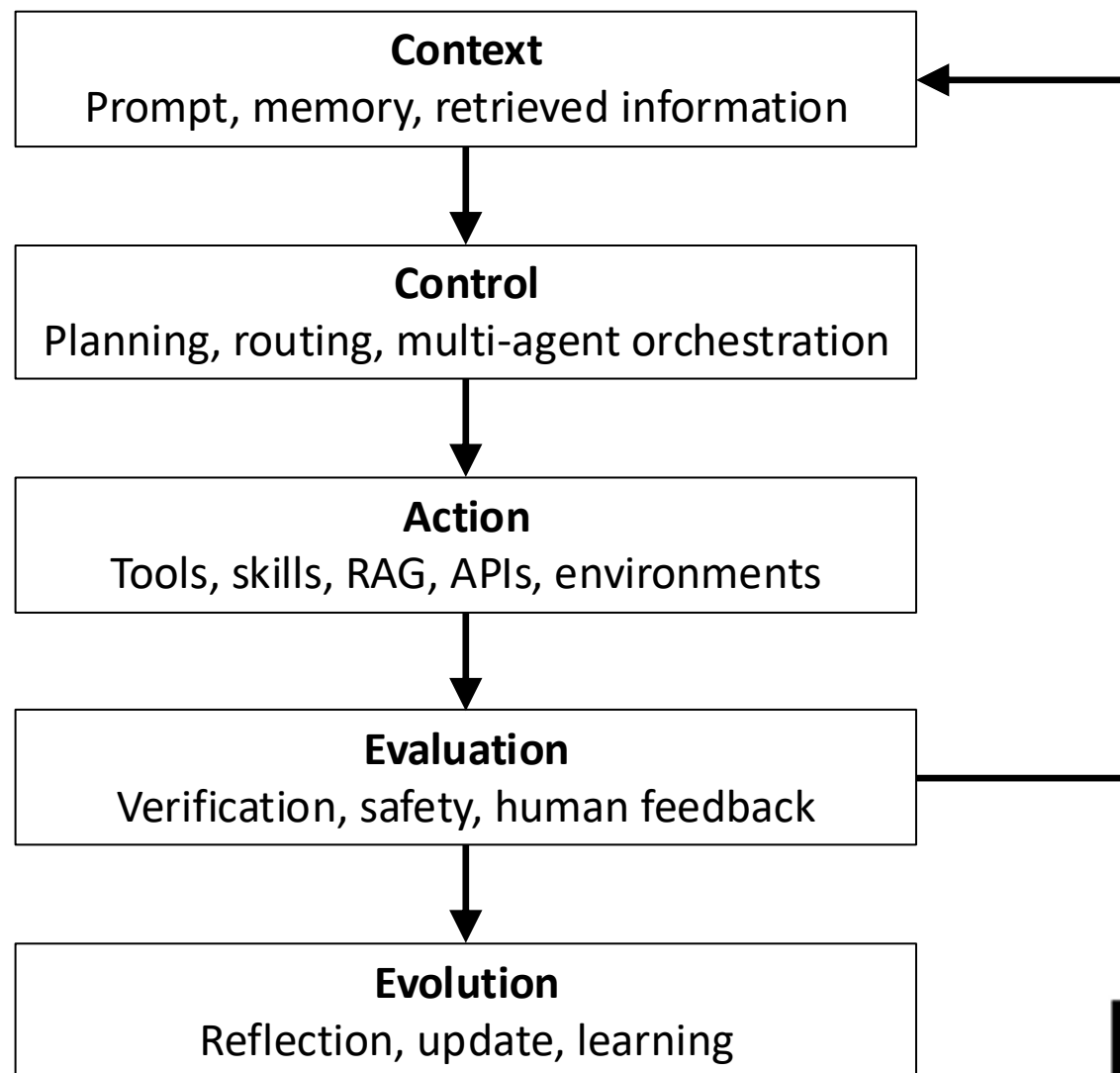
```
population = initialize_programs()
evaluate(population)

for generation in range(budget):
    parents = select(population)
    children = vary(parents) # mutation / crossover
    evaluate(children)
    population = retain(population, children)
```



Agents as Model + Harness

Harness: Code around the model that manages its context and execution flow to complete tasks



Glossary of what is Stored in Harness

Eg. In a coding agent harness

Episode/trajectory: state, action, next state, next action (e.g., good debugging examples)

Feedback: rewards, critiques, human preferences (e.g., user prioritizes clean code)

Facts: semantic knowledge (e.g., sorting function has certain complexity)

Tools: task-specific atomic (external) action (e.g., running python interpreter)

Skills: task-agnostic reusable procedure (e.g., use gdb to debug)

Artifacts: generated or optimized outputs (e.g., generated code)

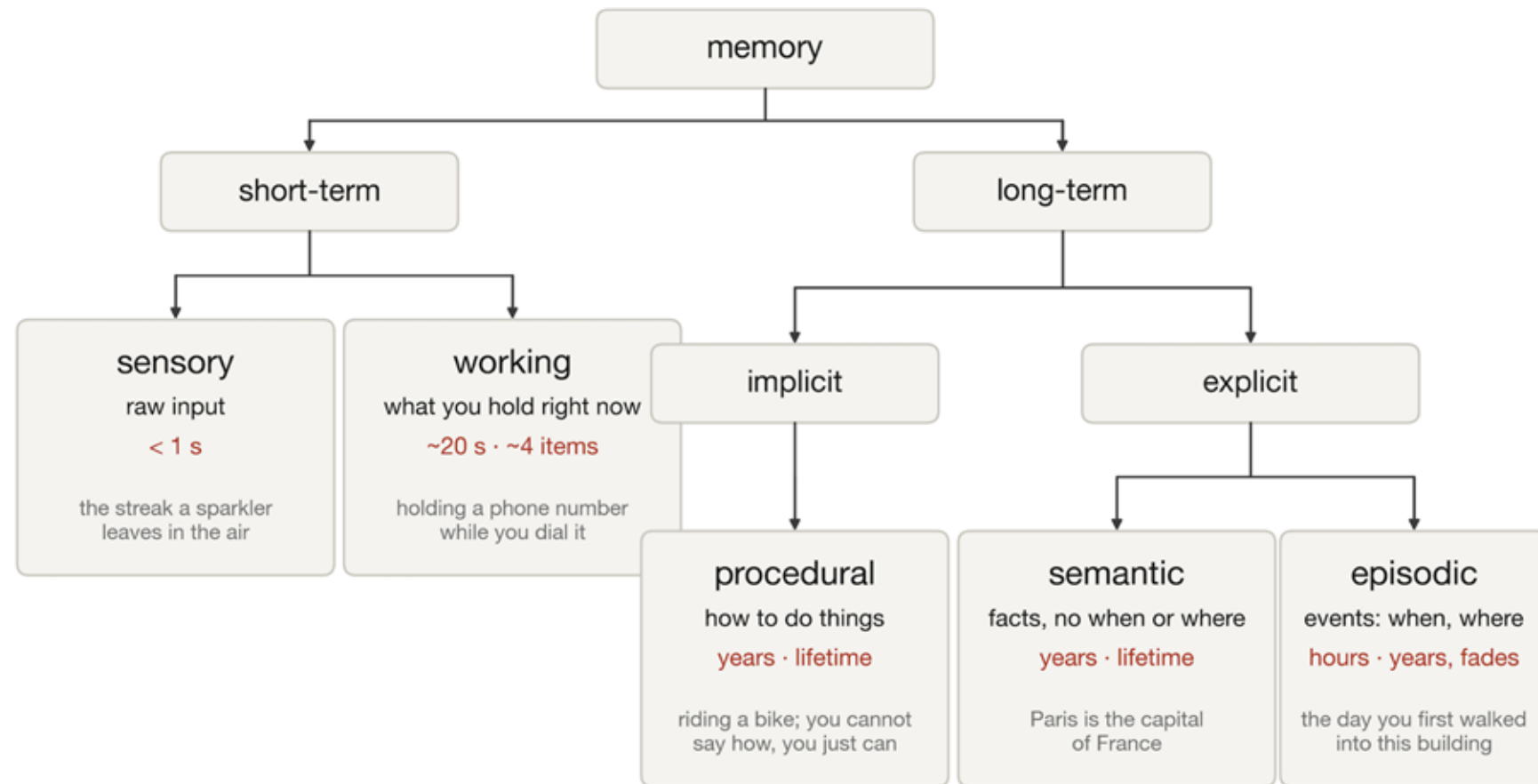
Plans: decomposed tasks and sub-goals (e.g., read requirements, write, modularize)

Experience: often used vaguely, encompasses trajectories, feedback, skills

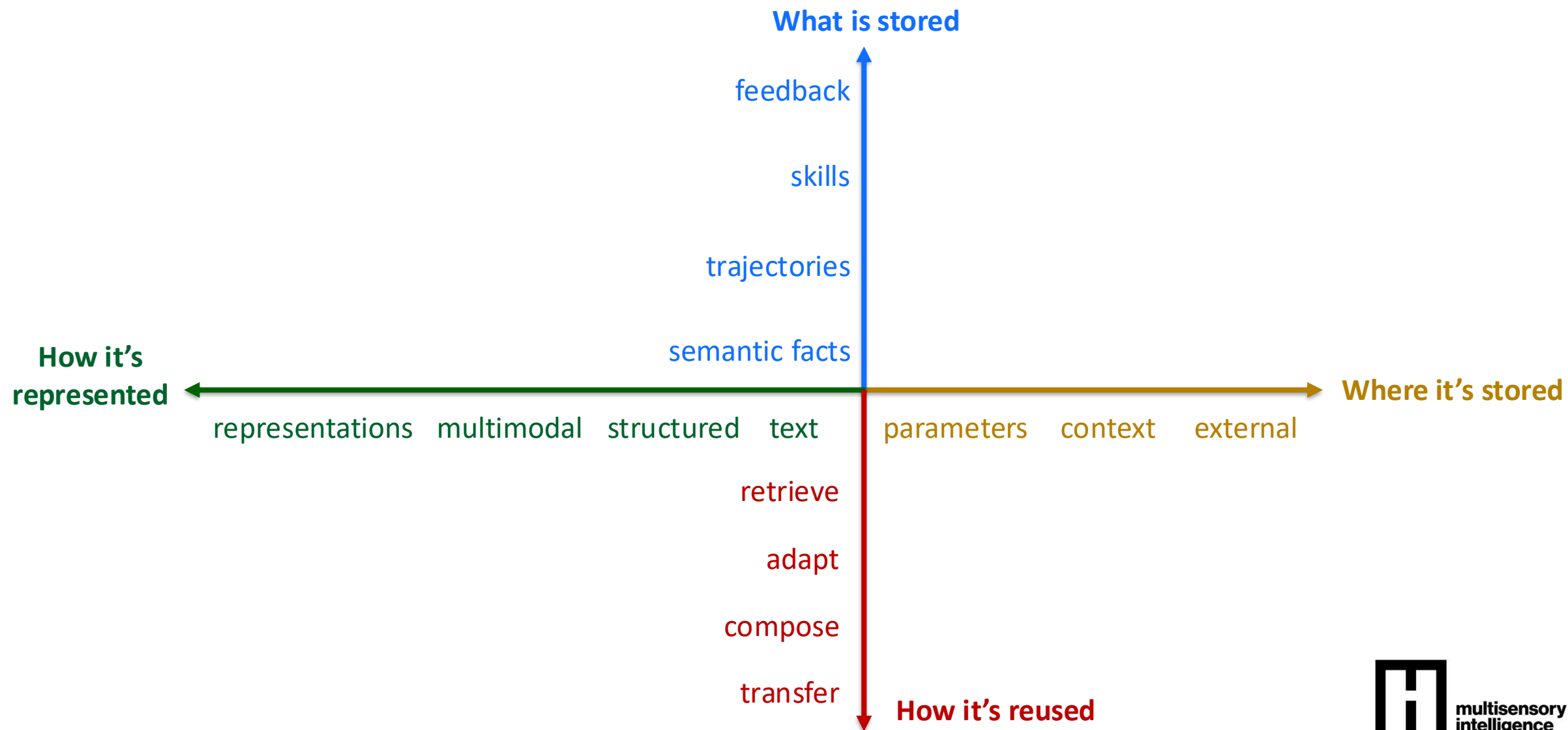
Harness Part 1: Memory

Definition of memory: Persistent state of an agent harness that stores information beyond the current context, enabling long-range and cumulative learning.

- Memory is part of context and harness
- Memory decides what to go into context
- Memory management is job of the harness



Overview of Memory



External Memory Systems

Reads and writes are function calls by the agent.

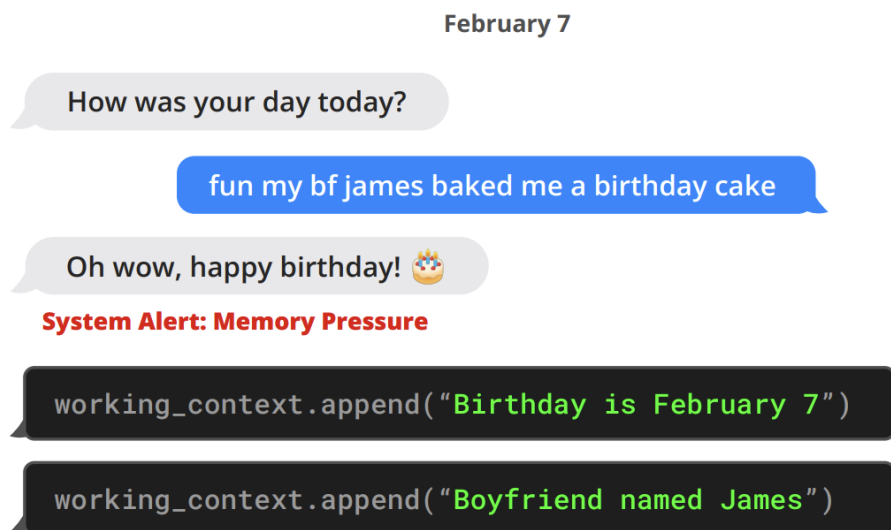


Figure 1. MemGPT (left) writes data to persistent memory after it receives a system alert about limited context space.

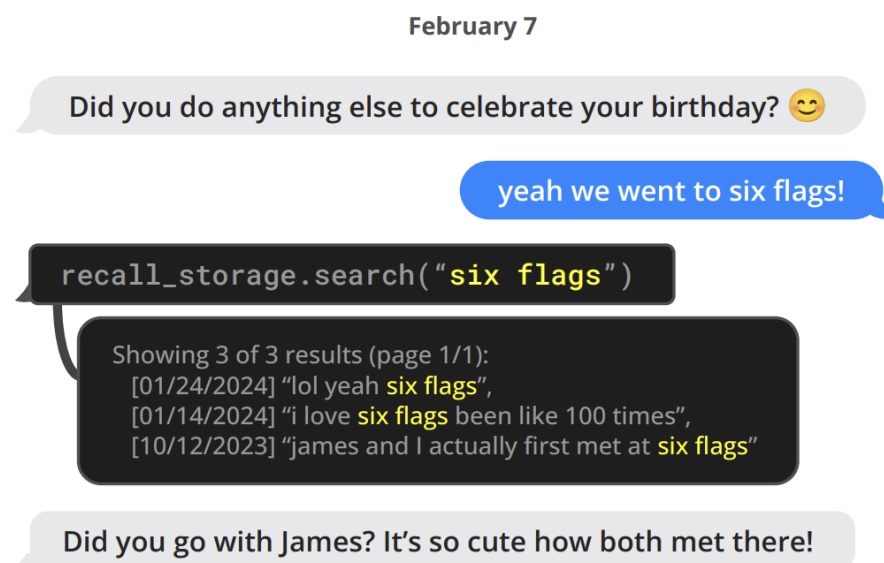
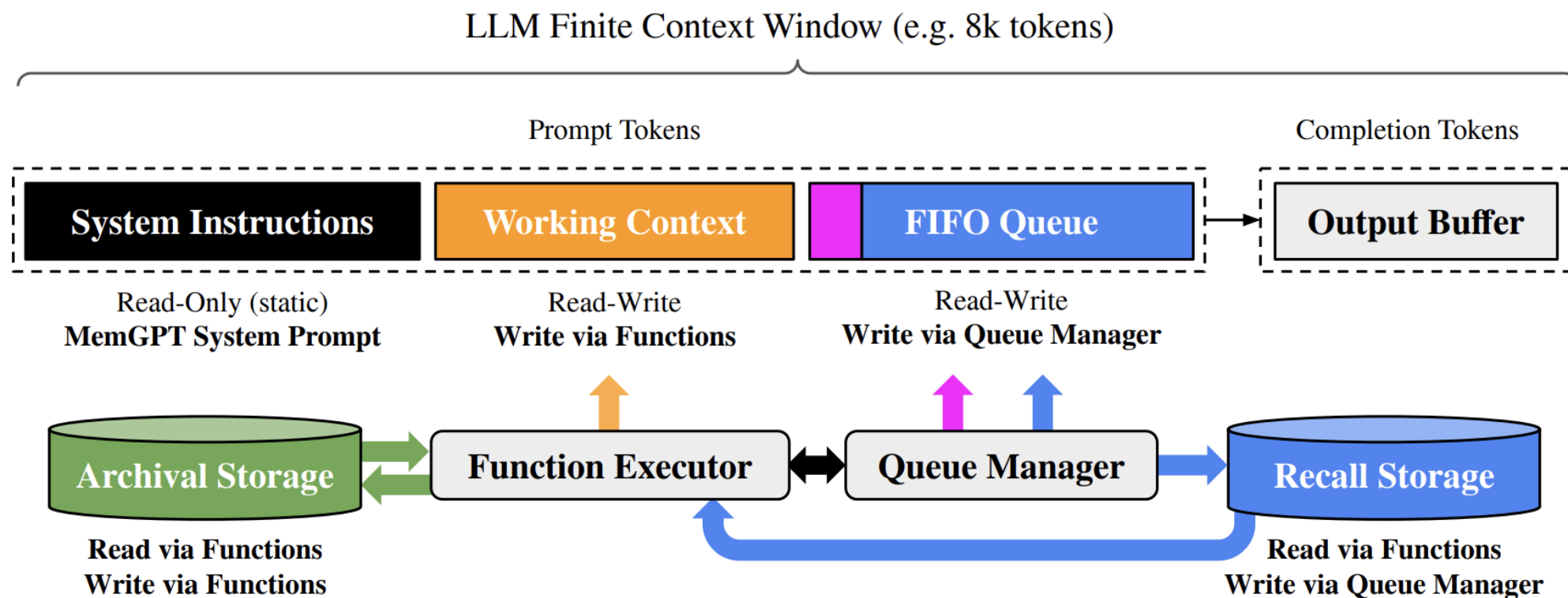


Figure 2. MemGPT (left) can search out-of-context data to bring relevant information into the current context window.

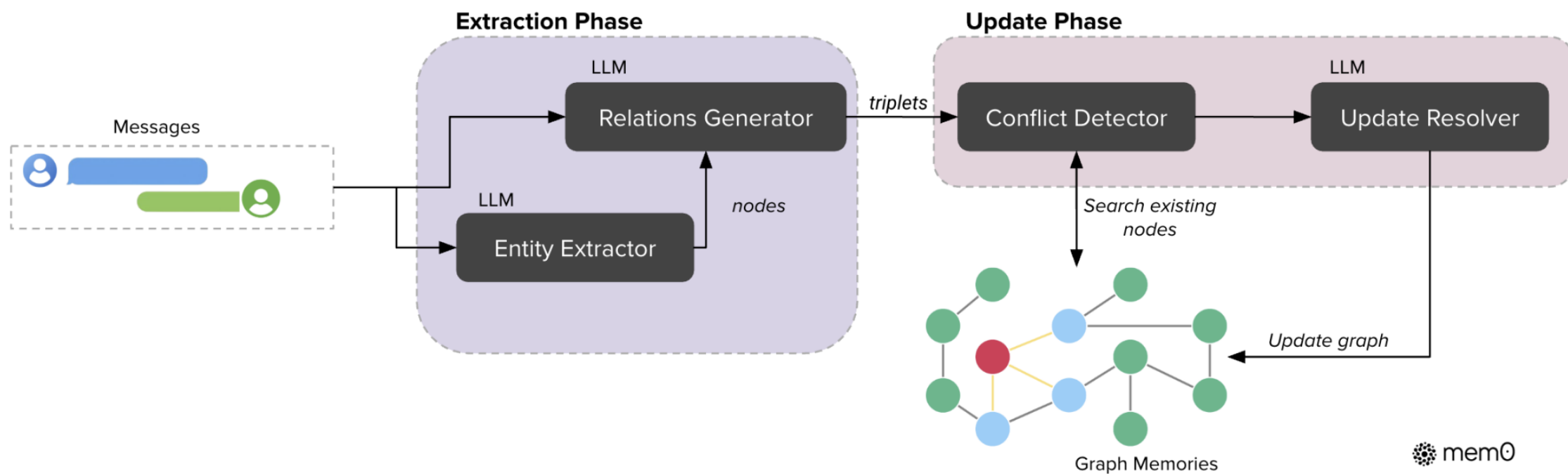
External Memory Systems

Reads and writes are function calls by the agent.



External Memory Systems

Memory is structured as a graph, converting messages into entity and relation triplets. Structured representations often used to represent knowledge, databases, symbolic states.



External Memory Systems

Add, update, delete, noop functions on memory

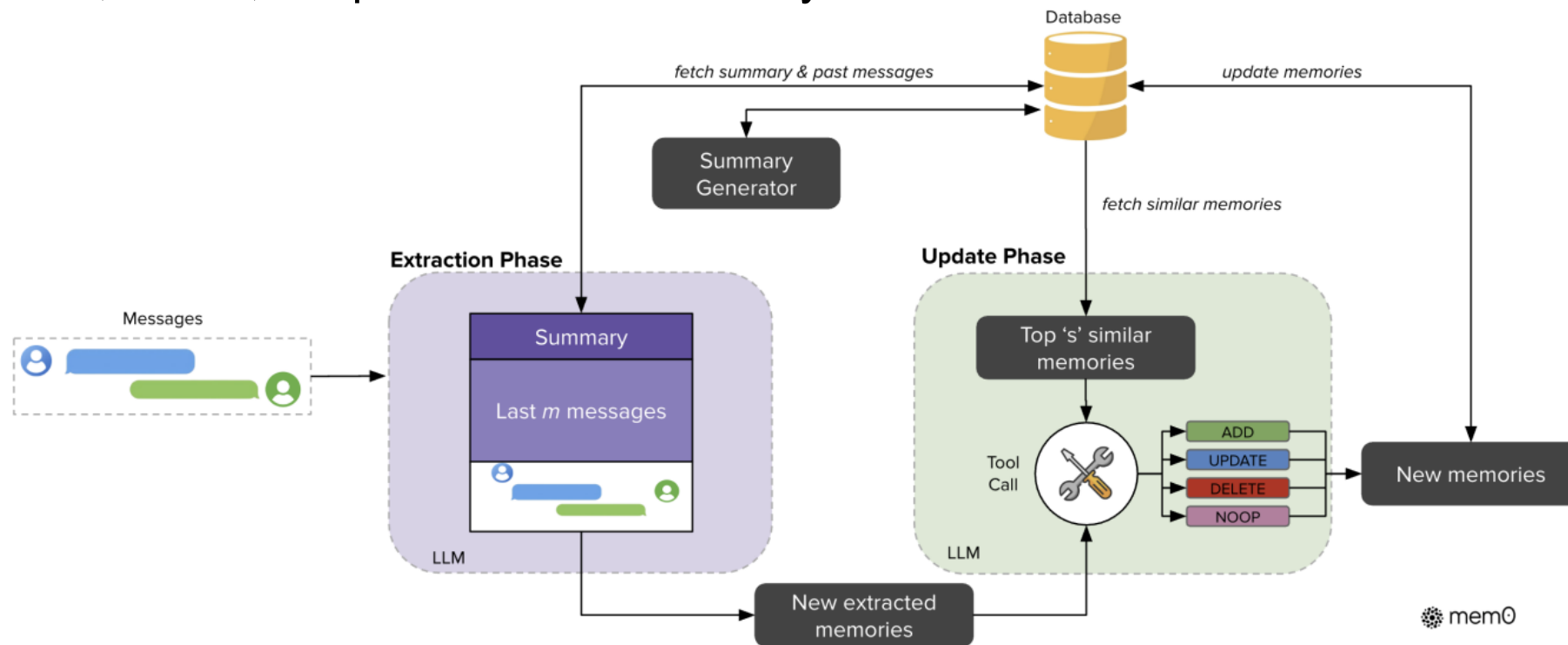
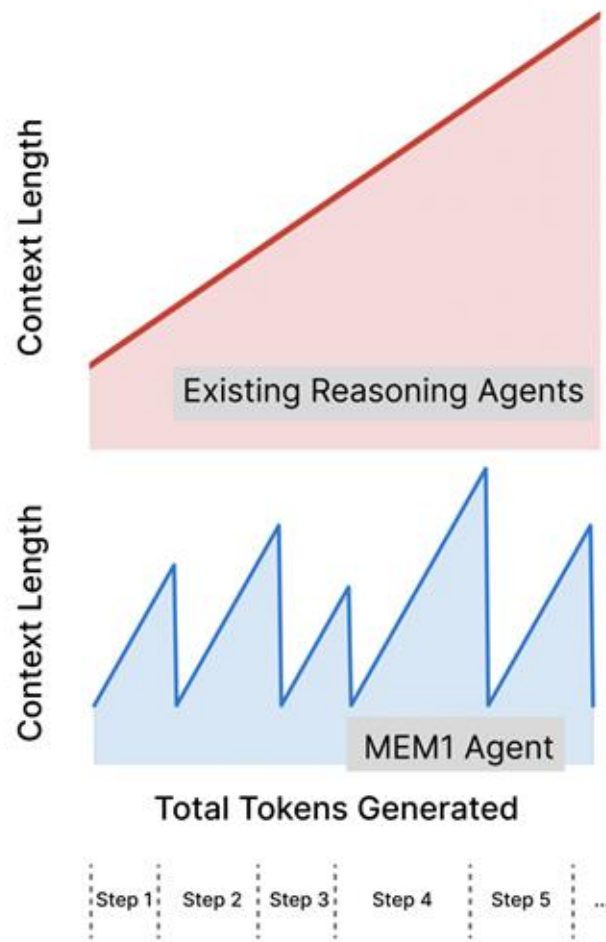


Figure 2: Architectural overview of the Mem0 system showing extraction and update phase. The extraction phase processes messages and historical context to create new memories. The update phase evaluates these extracted memories against similar existing ones, applying appropriate operations through a Tool Call mechanism. The database serves as the central repository, providing context for processing and storing updated memories.

Memory in Internal Context

<https://github.com/MIT-MI/MEM1>



Key problem: Growing reasoning context introduces complexity and forgetting challenges.

Proposal: Introduce an **internal state** that integrates **reasoning** and **memory**. At each step, the model **updates this internal state and discards all previous information**.

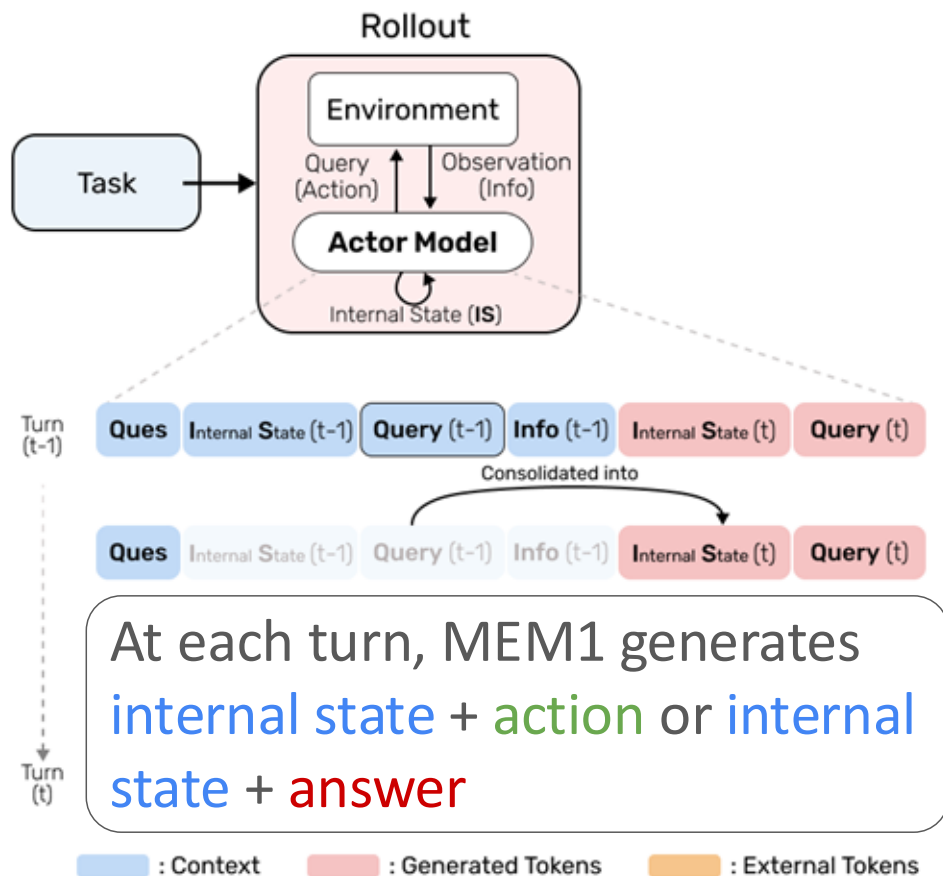
Advantages:

1. Enable near-constant memory
2. Support end-to-end optimization



MEM1 for Context Evolution

<https://github.com/MIT-MI/MEM1>

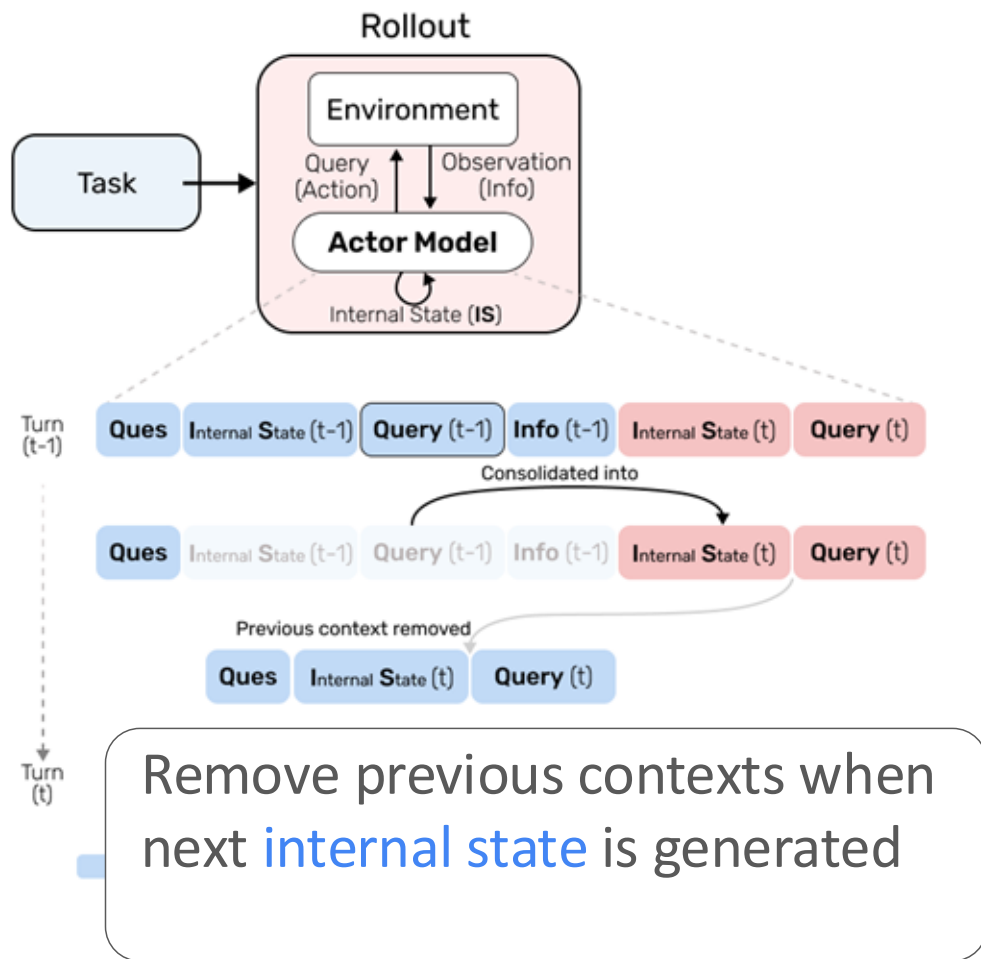


Key idea: Memory Consolidation

- MEM1 keeps context size constant by compressing past knowledge into the latest *Internal State*.
- Only the latest internal state, query, and info are in memory.

MEM1 for Context Evolution

<https://github.com/MIT-MI/MEM1>

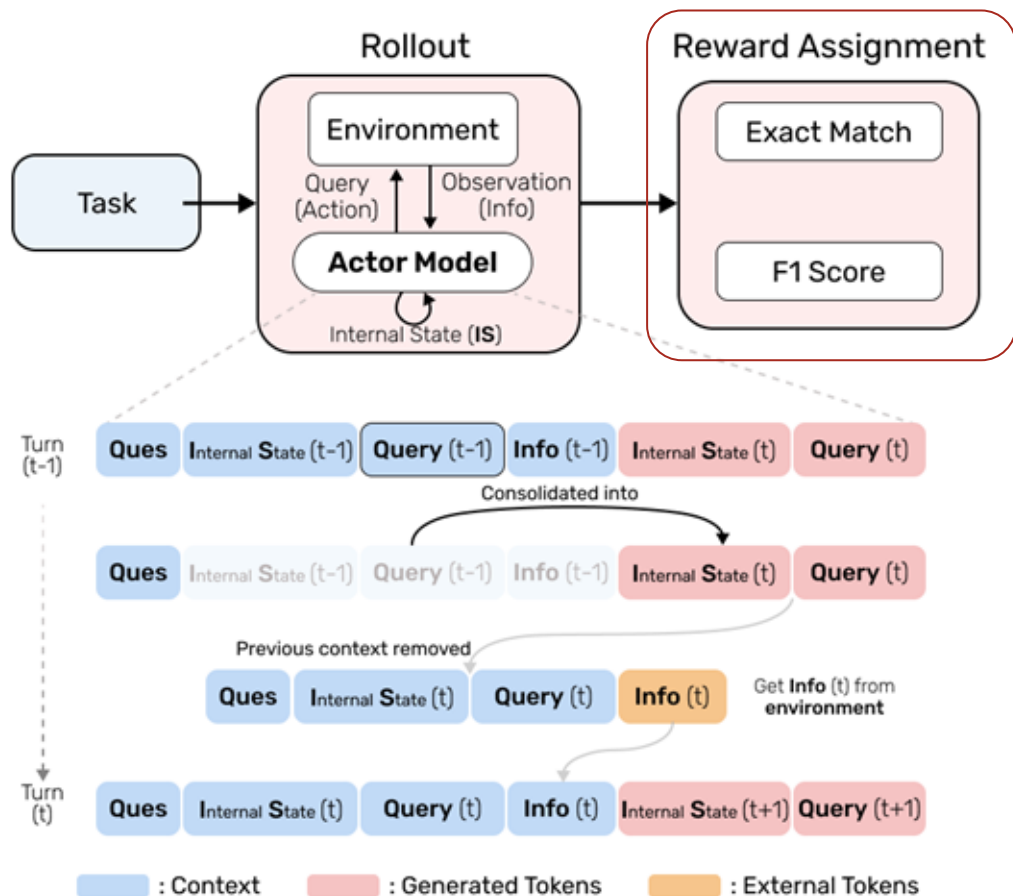


Key idea: Memory Consolidation

- MEM1 keeps context size constant by compressing past knowledge into the latest *Internal State*.
- Only the latest internal state, query, and info are in memory.

MEM1 for Context Evolution

<https://github.com/MIT-MI/MEM1>



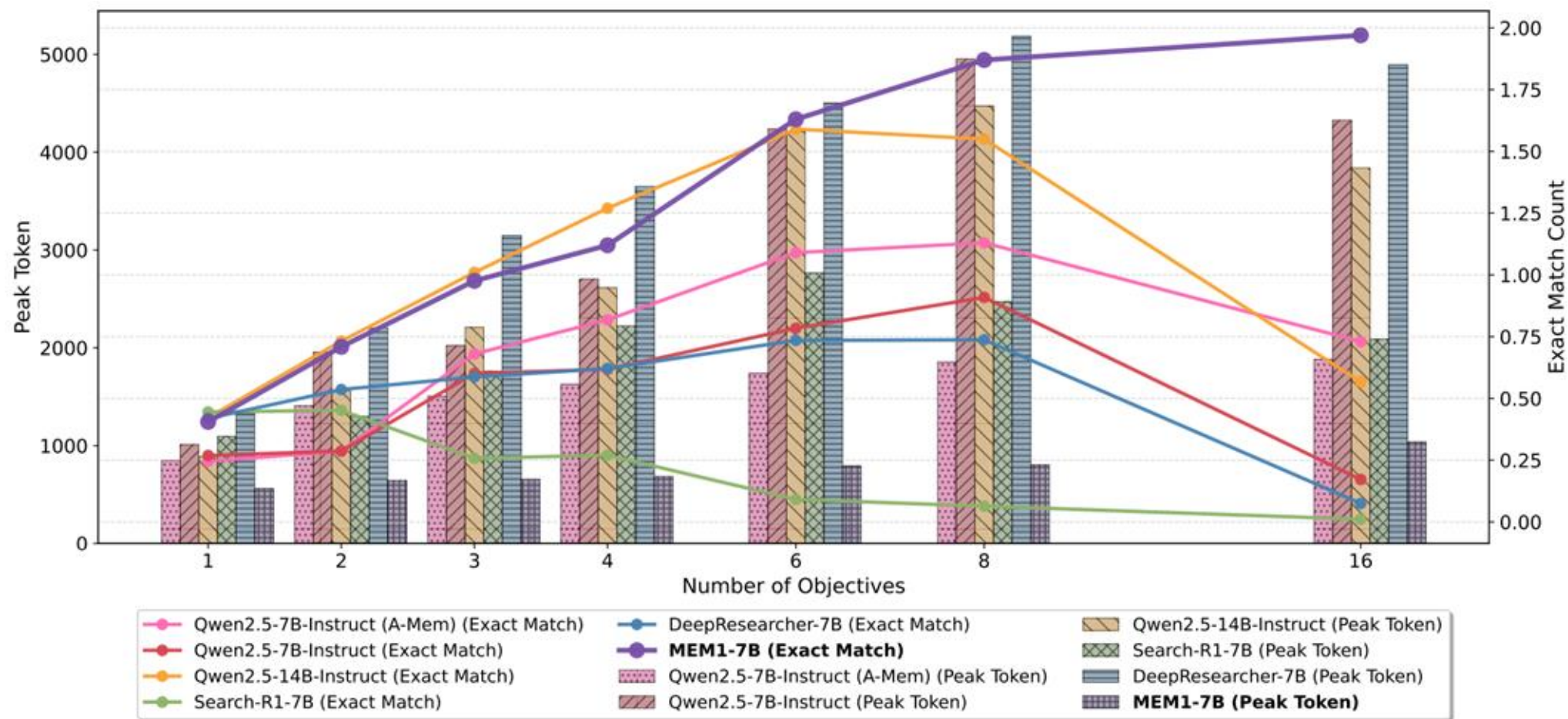
We use **answer** to calculate the outcome reward for each rollout trajectory

2 ways to use MEM1:

1. Direct prompting/in-context learning
2. Use reward to train model to compress

MEM1 Results: MultiHopQA

<https://github.com/MIT-MI/MEM1>



When # Questions $\uparrow$ $\rightarrow$ Performance : Baseline Methods $\downarrow$; MEM1 $\uparrow$
 Context : Baseline Methods $\uparrow$; MEM1 –

Memory Behavior

<https://github.com/MIT-MI/MEM1>



MEM1 learns to **store key extracted info** about each question separately

MEM1: ...

According to the provided information:

- They had 9 World Series appearances in Brooklyn.
- They have had 11 World Series appearances in Los Angeles.

Therefore, ... Now, to answer the second question, ...

From the provided information:

- The show originally ran on ABC from 2011 to 2017 for six seasons...

Memory Behavior

<https://github.com/MIT-MI/MEM1>

 MEM1 explicitly **updates its memory** before moving on to the next question

[Omitted History]

Env: [Search Results]

MEM1: : After searching for the information, we found out that Beniamino Di Giacomo was born in Porto Recanati. **Now we can add this information to our cumulative summary: Beniamino Di Giacomo was born in Porto Recanati.** To answer the second question, 'In what city was Tim Alexander born?', we need to perform a web search...

Memory Behavior

<https://github.com/MIT-MI/MEM1>

 MEM1 learns to skip one question after several searches without finding useful information.

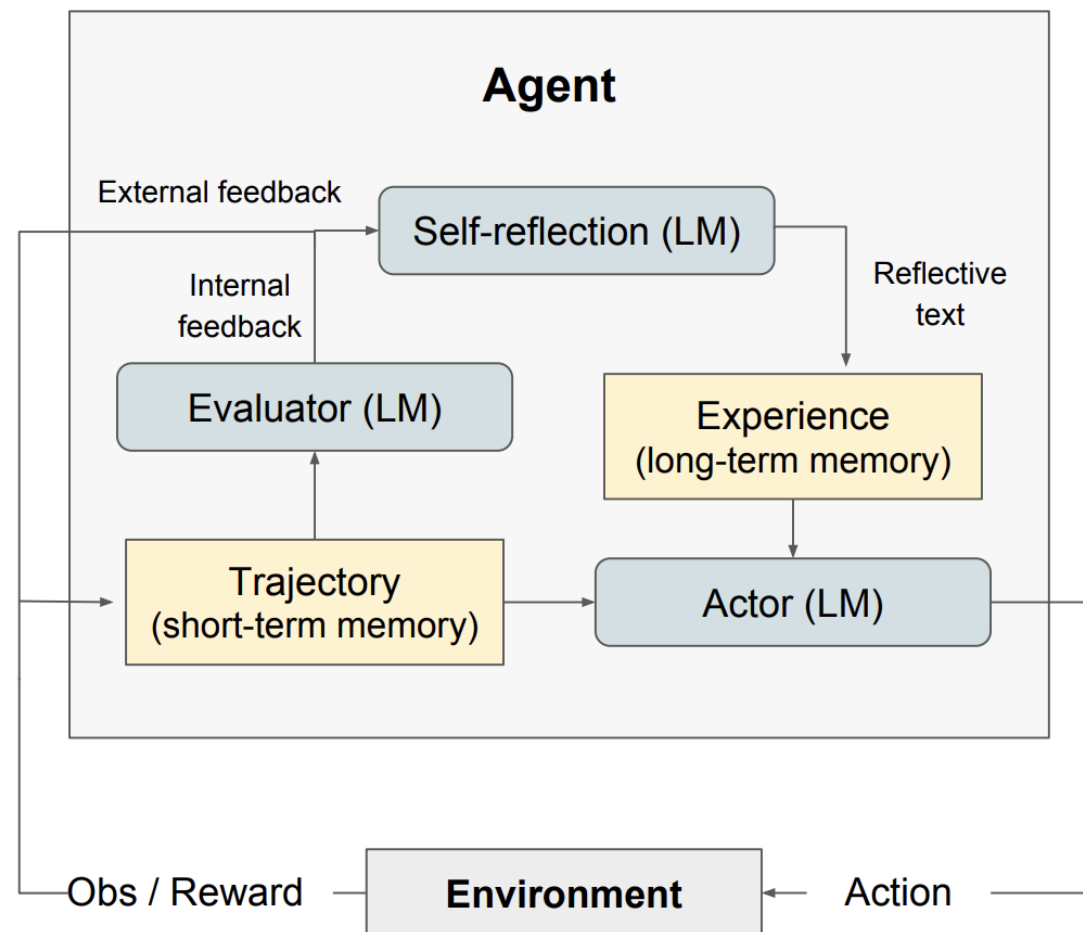
MEM1: Unfortunately, based on the information provided in the documents, I am unable to determine the name of the hotel designed by Veldon Simpson that was the largest in the world upon opening. Moving on to the second question, ...

Turning Memory into Useful Experience

Trajectories stored in short-term memory
Self-reflected experience in long-term memory

Model context is updated based on retrieval from memory

Combination of context + memory evolution



Turning Memory into Useful Experience

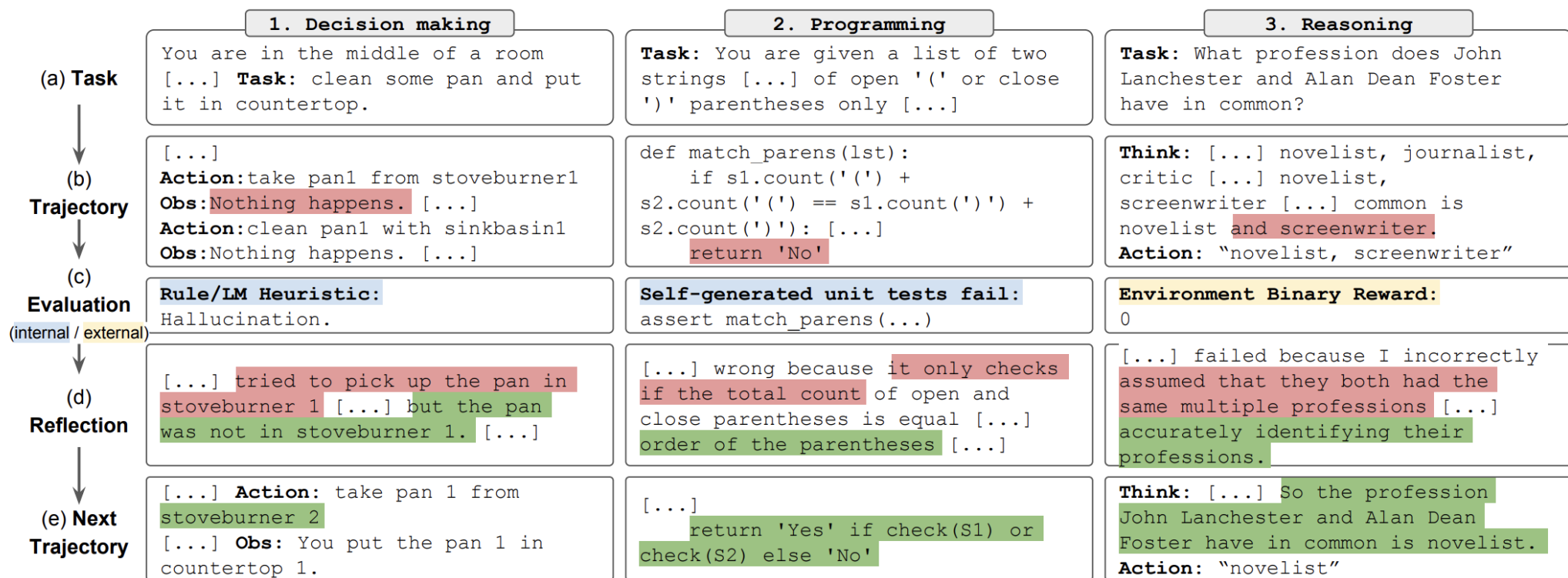


Figure 1: Reflexion works on decision-making 4.1, programming 4.3, and reasoning 4.2 tasks.

Turning Memory into Useful Experience

Building on Reflexion: trajectories into STM, reflections into LTM.

Adds **feedback** into memory: whether the experience was useful or not.

Trajectories, experience, feedback similar to the task given as few-shot examples.

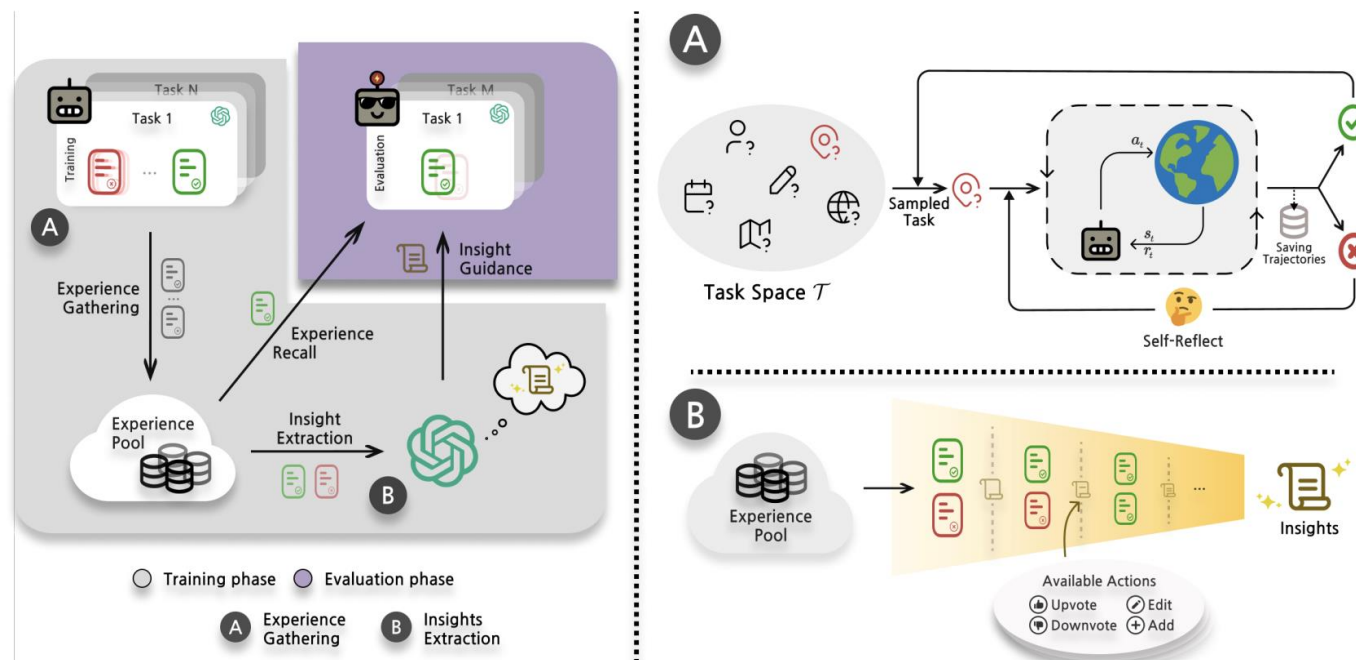


Figure 1: **ExpeL Agent Overview.** *Left:* ExpeL operates in three stages: (1) Collection of success and failure experiences into a pool. (2) Extraction/abstraction of cross-task knowledge from these experiences. (3) Application of the gained insights and recall of past successes in evaluation tasks. *Right:* (A) Illustrates the experience gathering process via Reflexion (Shinn et al. 2023), enabling task reattempt after self-reflection on failures. (B) Illustrates the insight extraction step. When presented with success/failure pairs or a list of L successes, the agent dynamically modifies an existing list of insights $\hat{l}$ using operations ADD, UPVOTE, DOWNVOTE, and EDIT. This process has an emphasis on extracting prevalent failure patterns or best practices.

Context Language Models

Wrapping up our discussion of context, memory, and harnesses, Context Language Models natively manage their own context by treating context as a file and allowing models to make unrestricted edits to this file.

Context Language Models (CLMs) generalize the append-only context transition of a standard LM to a model-controlled context transition. Standard LMs append model output to the current context:

$$c_{t+1} = c_t \oplus f_{\theta}^{\text{LM}}(c_t), \quad (1)$$

where $\oplus$ denotes concatenation. In contrast, CLMs delegate full responsibility for maintaining the context to the CLM itself, directly creating the next context:

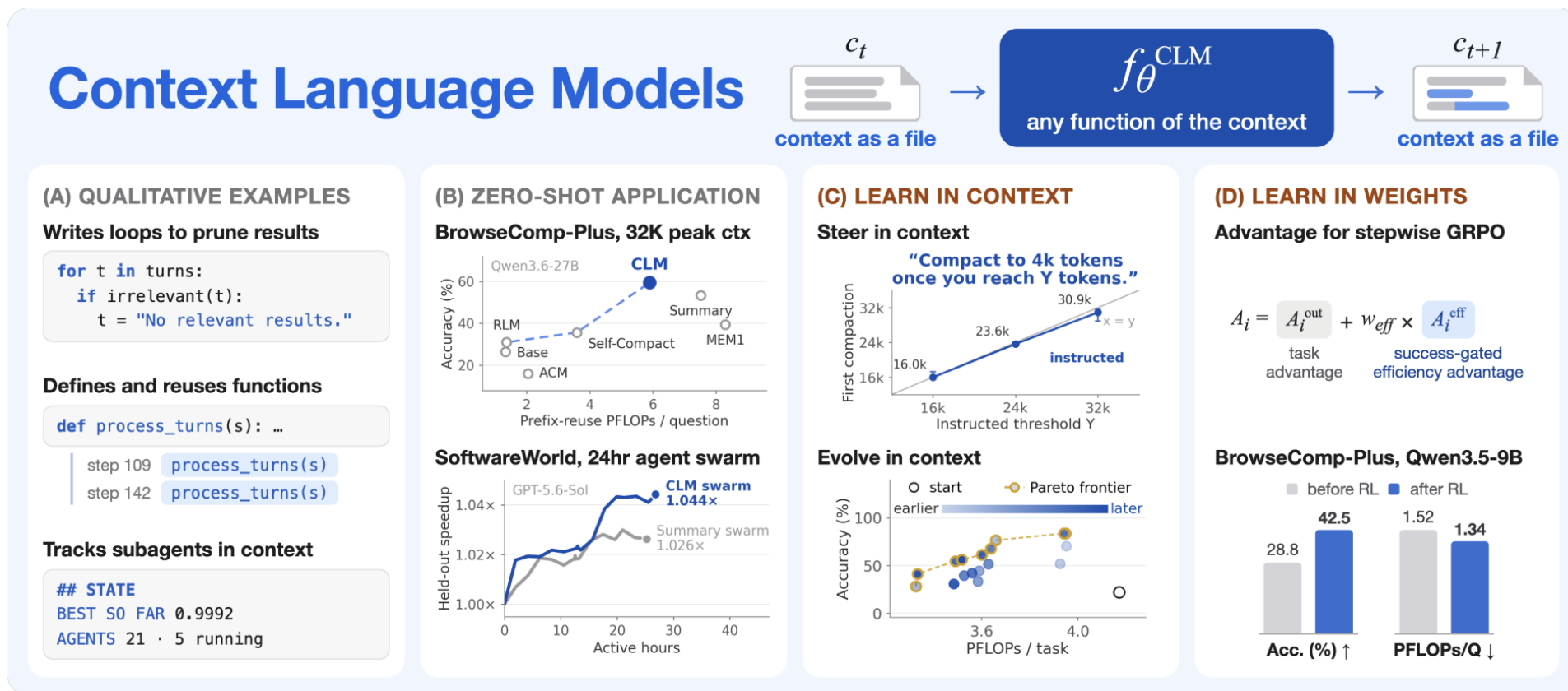
$$c_{t+1} = f_{\theta}^{\text{CLM}}(c_t), \quad (2)$$

Context-as-a-file implementation: LM's live context as a directly editable file with path provided in system prompt. LM edits this file using Bash commands, just like other files. Edits to context file are synchronized with the LM's context.

Multi-agent extensions: multiple context files can coexist and remain synchronized with their respective LLMs agents. Subagents can be initialized and terminated by creating and deleting additional context files.

Context Language Models

Wrapping up our discussion of context, memory, and harnesses, Context Language Models natively manage their own context by treating context as a file and allowing models to make unrestricted edits to this file.



Context Language Models

(a) CLM builds in-context scoreboards and trackers to orchestrate and monitor subagents with in-place editing.

```
open(p,"w").write("""[[CTX_TURN 1 role=assistant]]
### STATE — Erdős Minimum Overlap Problem (compact)
LEDGER TOP: 0.9992491 ...
AGENTS: 21 launched, 5 currently running ...
KEY FILES: /workspace/subctx_4/h_best_final.npy ...
FINDINGS: All methods plateau at 0.381157 ... """)
```

Erdős minimum overlap, step 455

```
new="""### ORCHESTRATOR STATE (compact)
Budget: 7/100 used. All 5 slots BUSY (subctx_0..4 ...
Dead ends: simple grid(0.822), hexagonal(0.9977) ...
Next: score my own candidates while workers run ... """)
```

Circle packing $N = 26$, steps 194, 198

(c) CLM writes for loops to remove past irrelevant search results or compact overly long outputs when creating a new view.

```
for t in turns[1:]: ...
    result += f'\n[[CTX_TURN search]]\nSearched: {m.group(1).strip}
↪ (). No relevant results.\n'
```

BrowseComp-Plus, step 1509

```
while i < len(lines): ...
    if len(body) > 500: ...
        if 'bcp_search' in body: ...
            result.append(f"[Searched: {query}]" )
        elif 'bcp_get_document' in body: ...
            result.append(f"[Retrieved doc {docid}]" )
        else:
            result.extend(body_lines)
```

BrowseComp-Plus, step 13

(b) CLM creates a new role alongside the original template roles for its own internal notes.

```
re.sub(r"\[CTX_TURN 4 .*?(?=\[CTX_TURN 16)",
      "" "[CTX_TURN 4 role=notes]"
STATUS: ... James Gallagher (docid=58939) ... """)
```

BrowseComp-Plus, step 40

(d) CLM defines and reuses a function to conveniently compact old results with a reference to its maintained note.

```
progress = "" "[Search progress: VERIFIED ... NEXT: ...]"
s = re.sub(..., progress, s)
```

```
def compact_turns(text):
    return re.sub(..., lambda m: m.group(0).split('\n')[0]
        + '\n[search results - see progress note]', text)
s = compact_turns(s)
```

BrowseComp-Plus, step 109

(e) CLM reproduces effective behaviors from existing baselines, preserving important facts and future TODOs in the summary.

```
re.sub(r"\[CTX_TURN 2.*",
      "[SUMMARY: ... Kader Asmal Excellence Award
launched 2011 by Mrs A Motshekga ...]" )
```

BrowseComp-Plus, step 20

```
new="""[EXPLORATION LEDGER - 86 scored attempts ...
Best score: 0.9931 (sum_radii=2.6177) from ...
UNTRIED IDEAS (priority order):
1. Gradient clipping norm=1.0 with 12k steps
2. Try lam=2200+uniform(0,2800) with 12k steps ... """)
```

Circle packing $N = 26$, step 332

Context Language Models

First experiments: zero-shot usage of CLMs

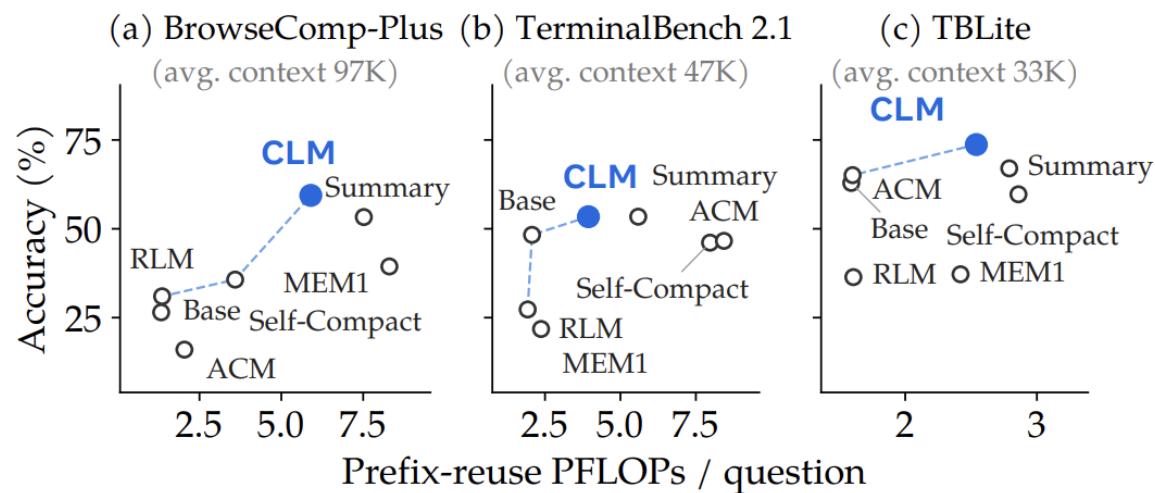


Figure 5 CLMs perform better than action-based and harness-defined baselines at lower cost on coding and deep research tasks. All methods use Qwen3.6-27B with a 32K context limit and a 100-turn cap. Blue dashed lines indicate the Pareto frontier.

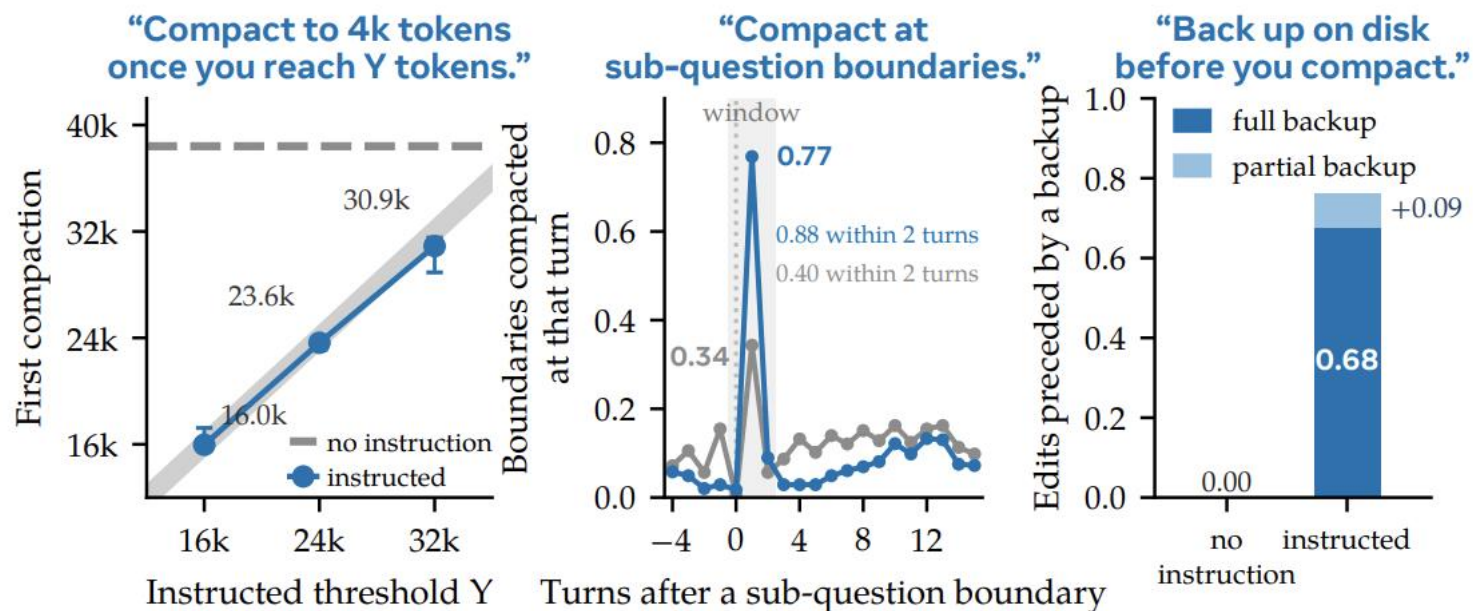
Table 1 CLMs outperform specialized OpenEvolve evolutionary workflows on mathematical optimization problems. Best-of-run scores with Claude 4.6 Sonnet and a 32K context limit, capped at 100 scored attempts or five hours. Arrows indicate the direction of improvement; OE stands for OpenEvolve and SA for subagents.

Method	Circle packing (↑)	Heilbronn (↑)	Min-max/min-dist (↑)	Erdős overlap (↓)
OE	2.541	0.03127	0.07690	0.38123
OE-Agent	2.525	0.03053	0.07724	0.38167
CLM	2.618	0.03653	0.07758	0.38094
CLM (SA)	2.636	0.03617	0.07758	0.38109

Context Language Models

Second experiments: fine-tuning CLMs via in-context learning, context evolution, or reinforcement learning

In-context instructions: $c_{t+1} = f_{\theta}^{\text{CLM}}(c_t; s)$.



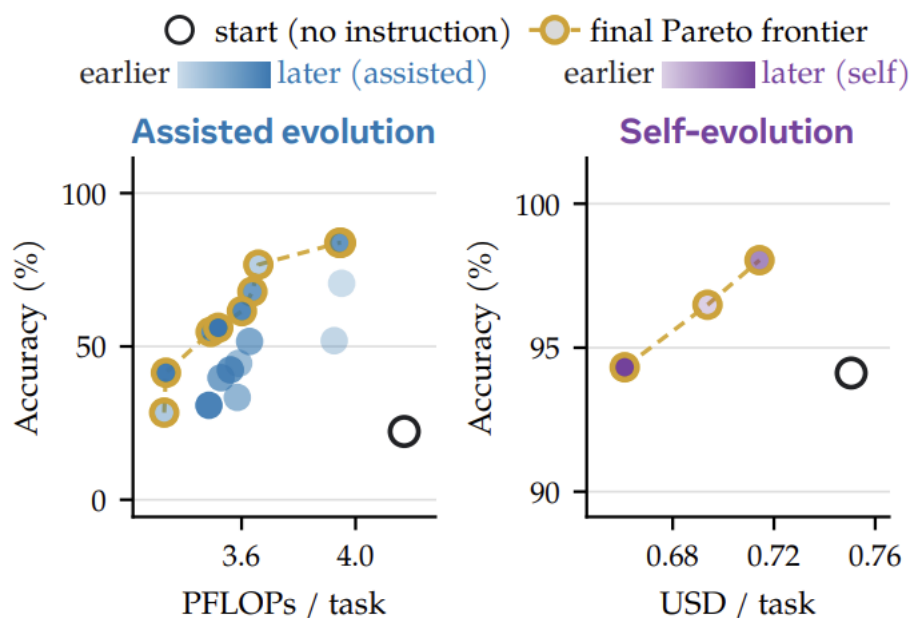
Context Language Models

Second experiments: fine-tuning CLMs via in-context learning, context evolution, or reinforcement learning

Context evolution: $s^* = \arg \max_s \mathbb{E}_{x \sim \mathcal{D}} [R(\tau(x; s))]$

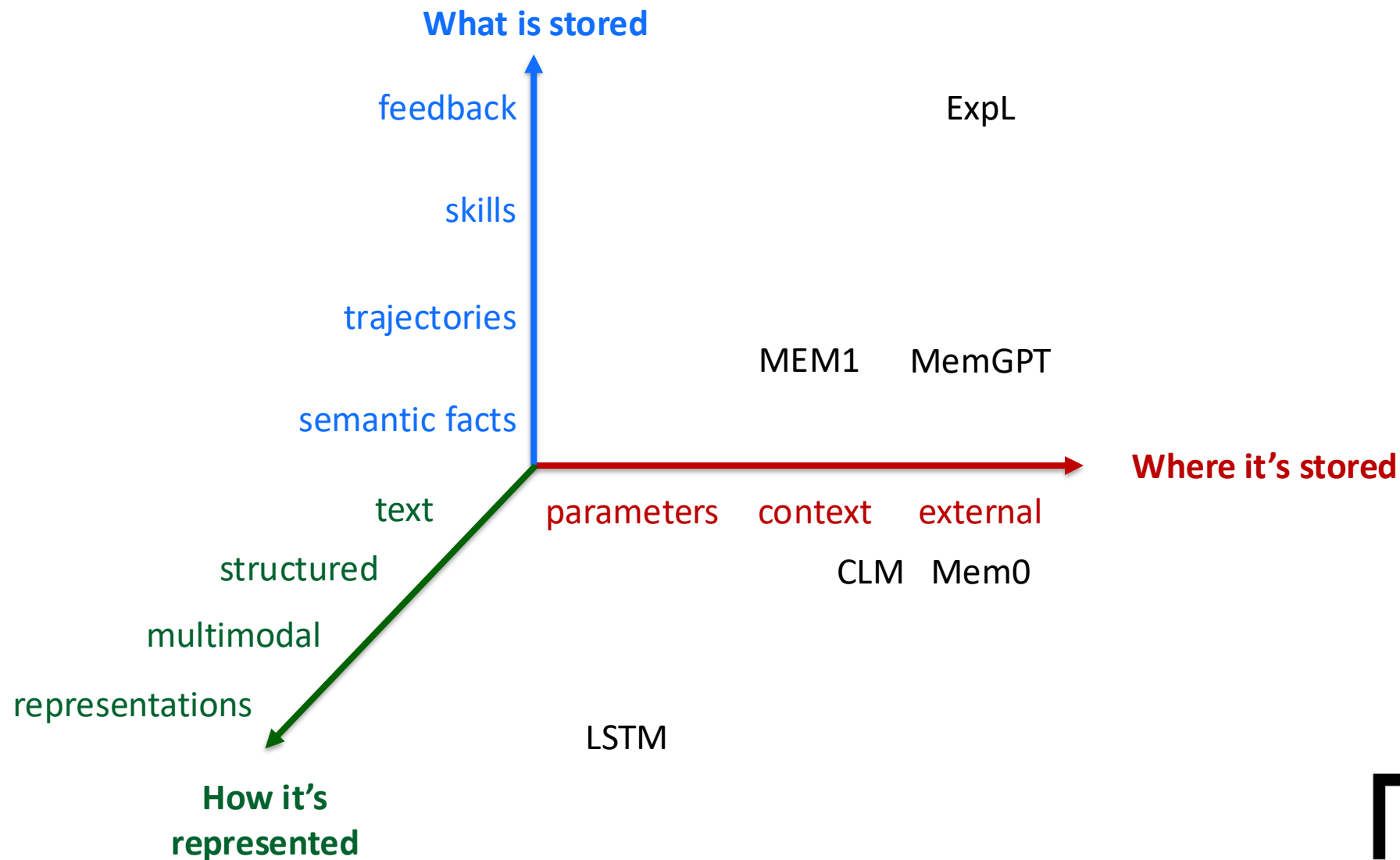
Reinforcement learning:

(reward is mix of task outcome and cache efficiency)



Method	Acc. (%) ↑	PFLOPs / Q ↓
Summary	34.7 → 42.1	4.01 → 2.19
CLM	28.8 → 42.5	1.52 → 1.34

Summary of Memory



Harness Part 2: Tools and Skills

Definition of tools: A task-specific atomic action that the agent can execute.

e.g., searching for a specific paper from a database and retrieving into context.

Definition of skills: A task-agnostic reusable procedure that an agent learns as part of completing a task. Often combines reasoning and multiple tool calls.

e.g., how to do a comprehensive search, clustering themes, writing a summary.

Tool Use

Category	Example Tools
 Knowledge access	<code>sql_executor(query: str) -> answer: any</code> <code>search_engine(query: str) -> document: str</code> <code>retriever(query: str) -> document: str</code>
 Computation activities	<code>calculator(formula: str) -> value: int float</code> <code>python_interpreter(program: str) -> result: any</code> <code>worksheet.insert_row(row: list, index: int) -> None</code>
 Interaction w/ the world	<code>get_weather(city_name: str) -> weather: str</code> <code>get_location(ip: str) -> location: str</code> <code>calendar.fetch_events(date: str) -> events: list</code> <code>email.verify(address: str) -> result: bool</code>
 Non-textual modalities	<code>cat_image.delete(image_id: str) -> None</code> <code>spotify.play_music(name: str) -> None</code> <code>visual_qa(query: str, image: Image) -> answer: str</code>
 Special-skilled LMs	<code>QA(question: str) -> answer: str</code> <code>translation(text: str, language: str) -> text: str</code>

Tool Use

Code is a meta-tool: code composes control flow, libraries, functions, and multiple operations in one action.

build-in functions → tools

Q: The bakers baked 200 loaves of bread ...
How many loaves of bread did they have left?

LM The bakers started with 200 loaves. They sold 93 in the morning ... The answer is 62.

```
loaves_baked, loaves_returned = 200, 6
sold_morning, sold_afternoon = 93, 39
answer = loaves_baked - loaves_sold_morning
        - loaves_sold_afternoon + loaves_returned
```

CodeLM

external libraries → tools

Q: The table shows how many... What is the
max number of vacation days across years?

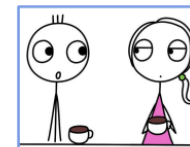
Year	2013	2014	2015
Vacation days	23	18	11

```
import pandas as pd
df = pd.DataFrame({"Year": [2013, 2014, 2015],
                  "Vacation days": [23, 18, 11]})
max_days = df["Vacation days"].max()
```

CodeLM

utility functions → tools

Q: Who is wearing the dress?



```
dress_box = locate_objects(image, "dress")
dress_region = crop_region(image, dress_box)
answer = visual_qa(dress_region,
                  question="Who is wearing the dress?")
```

CodeLM

Tool Calls via Code

Instruction: Determine the most cost-effective country to purchase the smartphone model "CodeAct 1". The countries to consider are the USA, Japan, Germany, and India.

Available APIs

[1] `lookup_rates(country: str) -> (float, float)`

[2] `convert_and_tax(price: float, exchange_rate: float, tax_rate: float) -> float`

[3] `estimate_final_price(converted_price: float, shipping_cost: float) -> float`

[4] `lookup_phone_price(model: str, country: str) -> float`

[5] `estimate_shipping_cost(destination_country: str) -> float`

LLM Agent using [Text/JSON] as Action

Think I should calculate the phone price in USD for each country, then find the most cost-effective country.

Action `Text: lookup_rates, Germany`
`JSON: {"tool": "lookup_rates", "country": "Germany"}`

Environment 1.1, 0.19

Action `Text: lookup_phone_price, CodeAct 1, Germany`
`JSON: {"tool": "lookup_phone_price", "model": "CodeAct 1", "country": "Germany"}`

Environment 700

Action `Text: convert_and_tax, 700, 1.1, 0.19`
`JSON: {"tool": "convert_and_tax", "price": 700, "exchange_rate": 1.1, "tax_rate": 0.19}`

Environment 916.3

[... interactions omitted (look up shipping cost and calculate final price) ...]

Action `Text: lookup_rates, Japan`
`JSON: {"tool": "lookup_rates", "country": "Japan"}`

[... interactions omitted (calculate final price for all other countries) ...]

Response The most cost-effective country to purchase the smartphone model is Japan with price 904.00 in USD.

Fewer Actions Required!

CodeAct: LLM Agent using [Code] as Action

Think I should calculate the phone price in USD for each country, then find the most cost-effective country.

Action

```
countries = ['USA', 'Japan', 'Germany', 'India']
final_prices = {}

for country in countries:
    exchange_rate, tax_rate = lookup_rates(country)
    local_price = lookup_phone_price("xAct 1", country)
    converted_price = convert_and_tax(
        local_price, exchange_rate, tax_rate
    )
    shipping_cost = estimate_shipping_cost(country)
    final_price = estimate_final_price(converted_price, shipping_cost)
    final_prices[country] = final_price

most_cost_effective_country = min(final_prices, key=final_prices.get)
most_cost_effective_price = final_prices[most_cost_effective_country]
print(most_cost_effective_country, most_cost_effective_price)
```

Control & Data Flow of Code Simplifies Complex Operations

Re-use `min` Function from Existing Software Infrastructures (Python library)

Environment 1.1, 0.19

Response The most cost-effective country to purchase the smartphone model is Japan with price 904.00 in USD.

Tool Calls via Code

Tools via code provided best success rate with the fewest interaction turns.

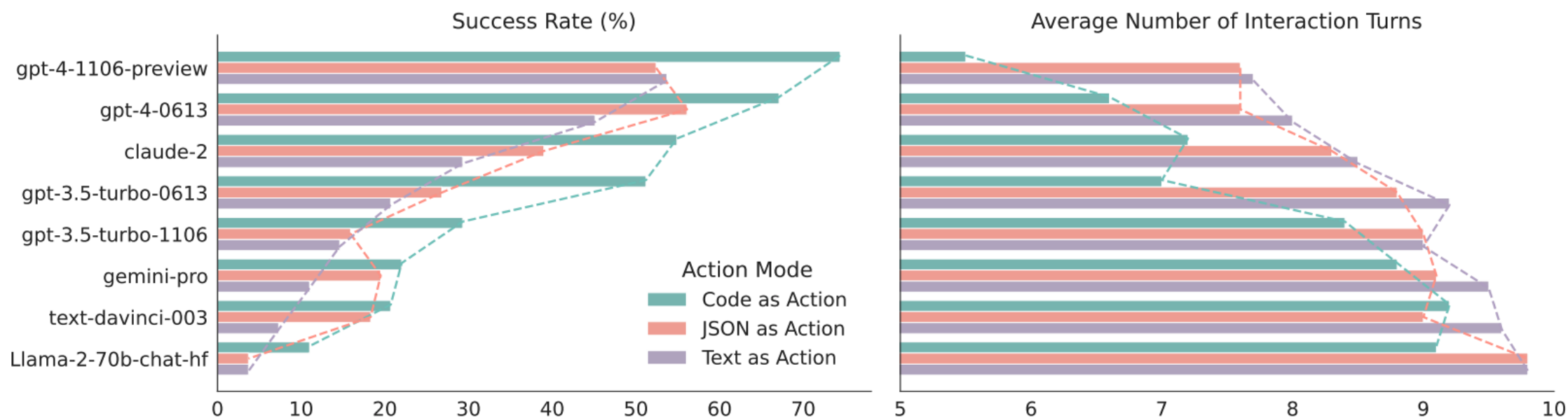


Figure 1: Comparison between CodeAct and Text / JSON as action. **(top)** Illustrative example comparing different actions. **(bottom)** Quantitative results on M³ToolEval (§2.3).

Learning Single-step Tool Calls

Training a policy for single-step tool calls. Main contribution: large dataset of API tool uses, model tries and filters different tool uses, and trains on successful ones.

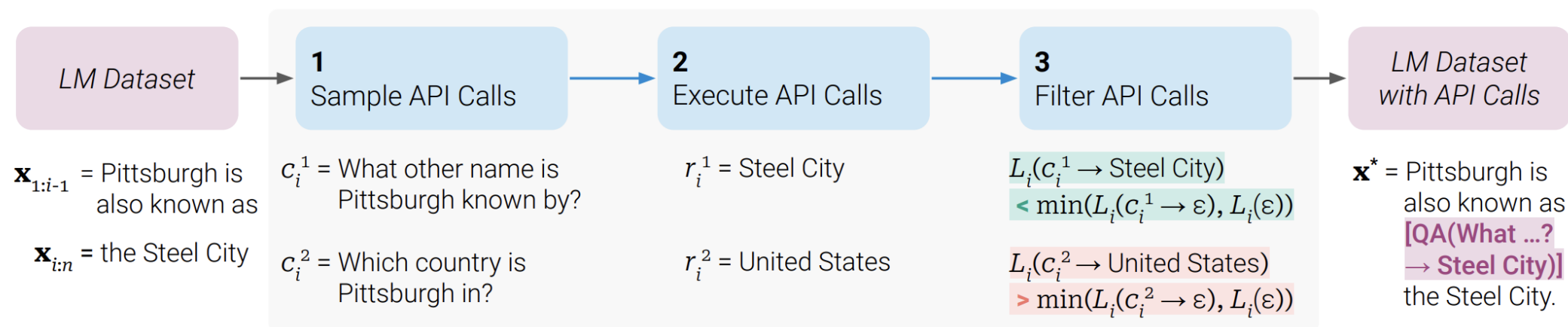


Figure 2: Key steps in our approach, illustrated for a *question answering* tool: Given an input text $\mathbf{x}$, we first sample a position i and corresponding API call candidates $c_i^1, c_i^2, \dots, c_i^k$. We then execute these API calls and filter out all calls which do not reduce the loss L_i over the next tokens. All remaining API calls are interleaved with the original text, resulting in a new text $\mathbf{x}^*$.

Multi-step Reasoning with Tools

ToolLLM extends single-step tool calls to multi-step reasoning + tool use and significantly scales the number of tools supported. RapidAPI dataset of tool hierarchies.

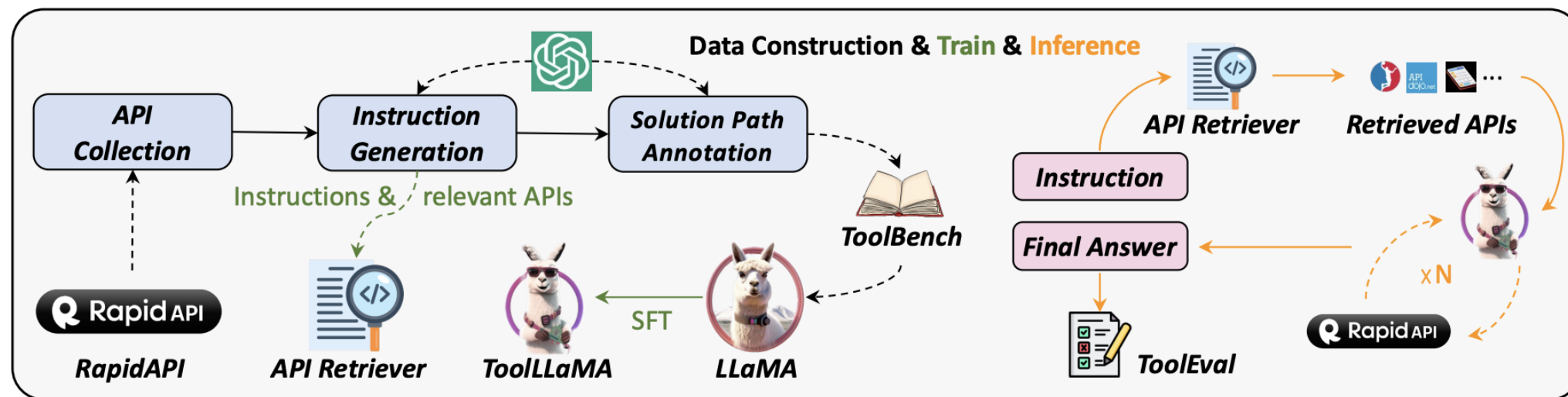
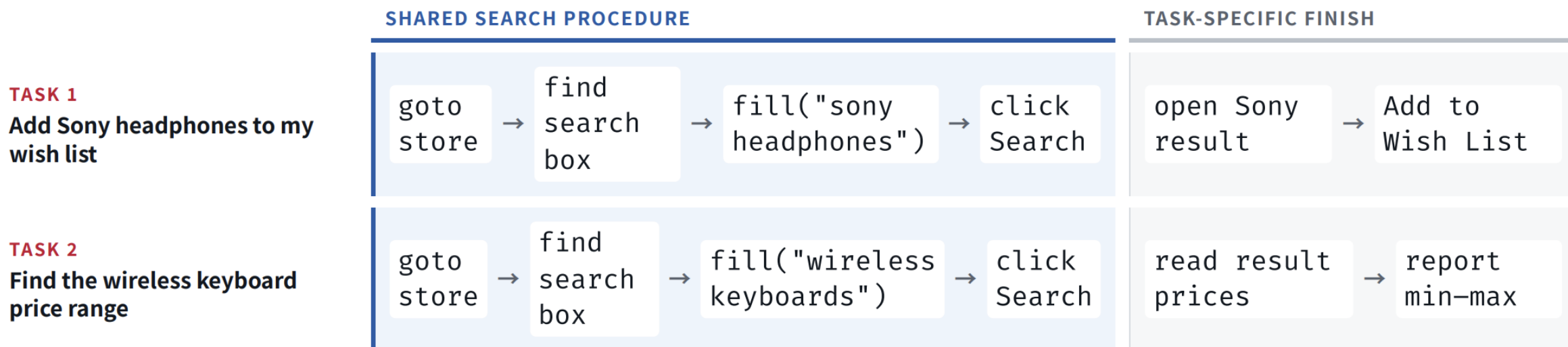


Figure 1: Three phases of constructing ToolBench and how we train our API retriever and ToolLLaMA. During inference of an instruction, the API retriever recommends relevant APIs to ToolLLaMA, which performs multiple rounds of API calls to derive the final answer. The whole reasoning process is evaluated by ToolEval.

Skills from Shared Task Structure

Tools: A task-specific atomic action that the agent can execute. e.g., searching for a specific paper from a database and retrieving into context.

Skills: A task-agnostic reusable procedure as part of completing a task. Often combines reasoning and multiple tool calls. e.g., a comprehensive literature search, clustering themes, writing a summary.



Lots of structure recurs in agent tasks.

Skills as Text vs Skills as Code

Text + examples

Task: Search for gaming accessories within a date range

Action trajectory:

```
click(1274) # Navigate to the Video Games category
fill(473, {search_terms}) # Enter search terms
                        including product name and year
click(478) # Execute the search
```

- + **Flexible:** a hint the agent interprets and adapts to the page in front of it
- **Manual:** the agent still issues every low-level action itself, copying and modifying

Code

```
def search_product(search_box_id: str, query: str):
    """Search for a product using the search box.
    Args:
        search_box_id: ID of the search input field
        query: Search query string to enter
    Returns:
        None
    Examples:
        search_product('595', 'sony bluetooth headphones')
    """
    click(search_box_id)
    fill(search_box_id, query)
    keyboard_press('Enter')
```

`search_product('595', 'Macbook')` **Usable as a tool**

- + **Testable:** run it before storing it
- + **Hierarchical:** skills call skills
- + **Efficient:** one call replaces many model steps
- **Rigid:** does exactly what it says, even on a page that has changed

Skills as Abstract Classes in Code

PolySkill Abstract Class

```
# The Abstract Domain Class, defining a "schema" for shopping sites.
class AbstractShoppingSite:
    def search_product(self, query: str):
        """Searches for a product."""

    def add_to_cart(self, item_id: str, quantity: int):
        """Adds a specified item to the shopping cart."""

    def checkout(self):
        """Initiates the checkout process."""

    ## -----
    ## Compositional Skills
    ## -----

    def find_and_add_to_cart(self, query: str, item_id: str):
        """Searches for a product and adds it to the cart."""
        self.search_product(query)
        self.add_to_cart(item_id)

    def purchase_item(self, query: str, item_id: str):
        """Finds a specific product, adds it to the cart, and starts
checkout."""
        self.find_and_add_to_cart(query, item_id)
        self.checkout()
```

PolySkill Implementation

```
class AmazonWebsite(AbstractShoppingSite):
    def search_product(self, query: str):
        click(search_bar_id)
        fill(search_bar_id, query)
        keyboard_press('Enter')

    def add_to_cart(self, item_id: str, quantity: int):
        click(add_to_cart_button_id)

    def checkout(self):
        """NOTE: Implements checkout as a composite action: first
cart, then clicks checkout."""
        self.view_cart()
        click(checkout_button_id)
```



```
class TargetWebsite(AbstractShoppingSite):
    def search_product(self, query: str):
        fill(search_bar_id, query)
        click(search_button_id)

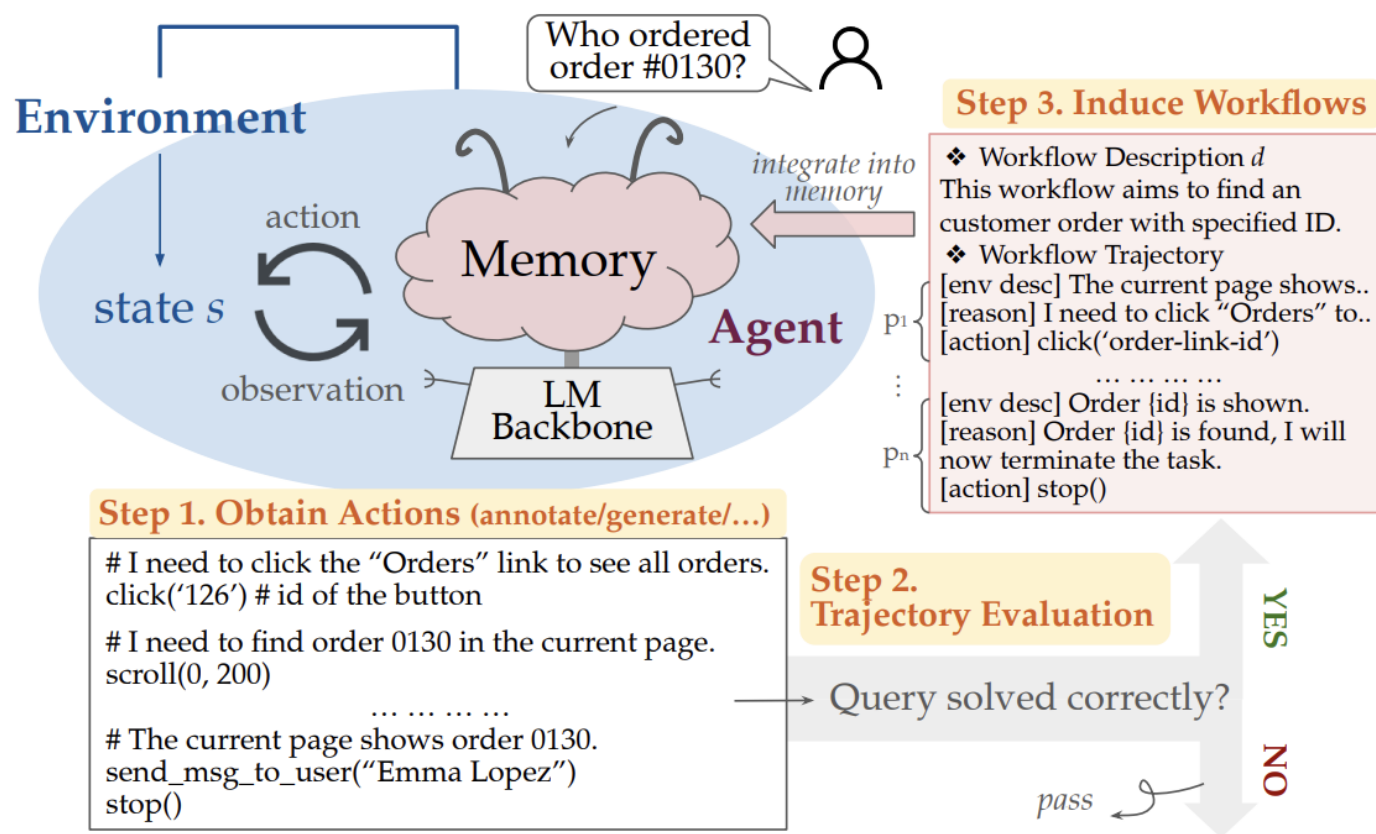
    def add_to_cart(self, item_id: str, quantity: int = 1):
        click(add_button_id)
        fill(quantity_bar_id, quantity)
        click(add_to_cart_button_id)

    def checkout(self):
        click(checkout_button_id)
```



Skill Learning from Memory Trajectories

Common workflows (i.e., sequences of skills), can be learned from past successful and failed trajectories, and stored in memory.



Skill Learning from Memory Trajectories



INDUCTION PROMPT FROM AGENT WORKFLOW MEMORY

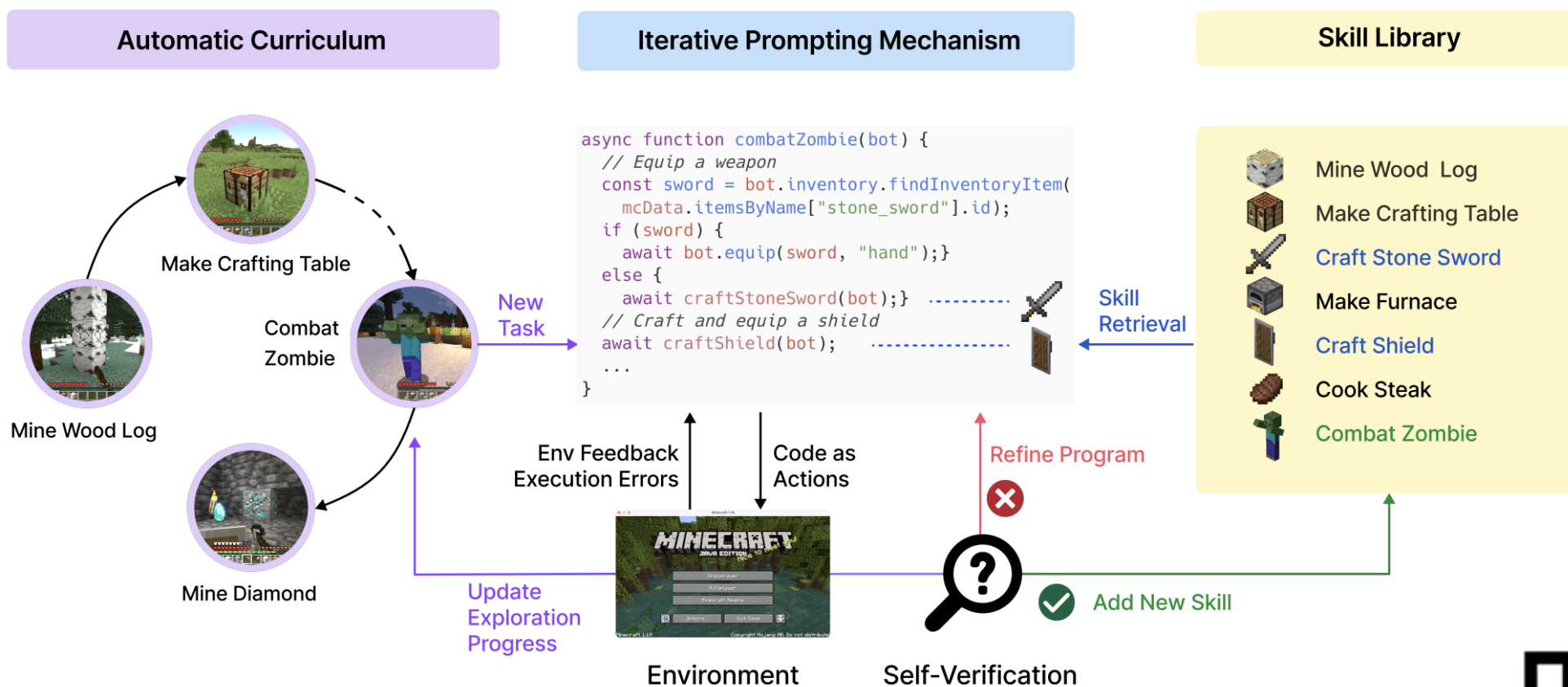
Given a list of web navigation tasks, your task is to extract the common workflows.

Each given task contains a natural language instruction, and a series of actions to solve the task. You need to find the repetitive subset of actions across multiple tasks, and extract each of them out as a workflow.

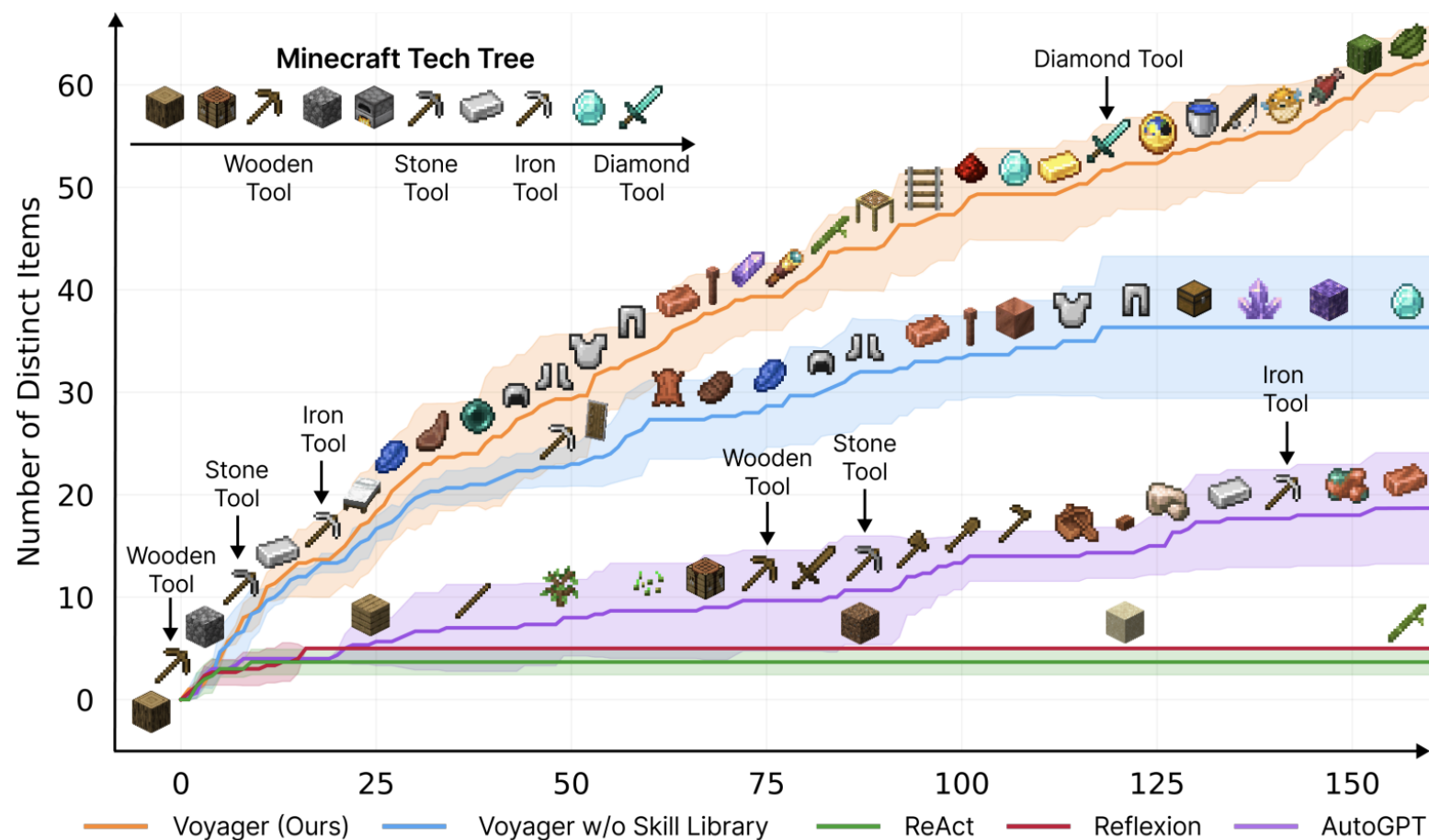
Each workflow should be a commonly reused sub-routine of the tasks. Do not generate similar or overlapping workflows. Each workflow should have at least two steps. Represent the non-fixed elements (input text, button strings) with descriptive variable names as shown in the example.

Skill Library

Combines several ideas: automatic curriculum generation, environment interaction and feedback, tool use, skill induction, stored in library (memory).



Skill Library



Meta-Harness

LLMs optimizing text/code often use short summaries, scalar scores, some past candidates.

Harnesses are more complex!
Meta-Harness proposer uses a filesystem with full source code, scores, and prior execution traces.

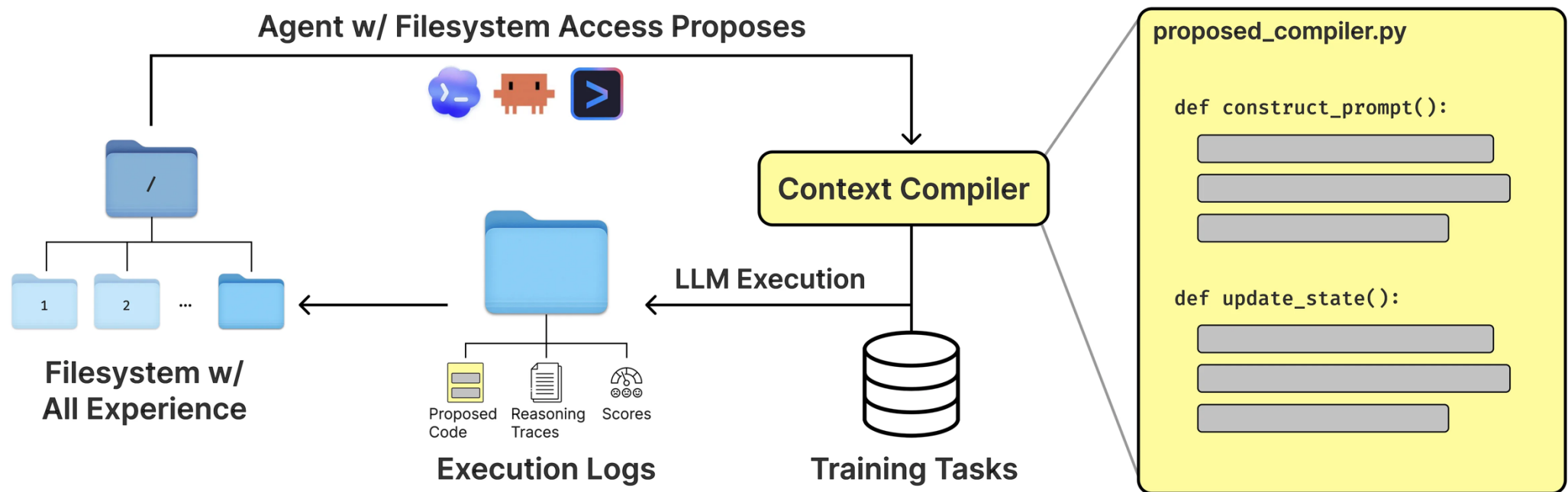
Claude Code proposer reads via grep & cat (10M tokens of feedback vs 26K for prior methods).
Proposer can trace a failure back to the specific harness decision.

Method	History	Log content	Mtok/iter ↑
Self-Refine	Last	output + self-generated critique	0.001
OPRO	Window	past (solution, score) pairs	0.002
TextGrad	Last	LLM textual gradient	0.015
MIPRO	Summary	bootstrapped program traces	0.003
AlphaEvolve	Window	program database + eval. scores	0.022
GEPA	Summary	rollout traces (reasoning + tools)	0.008
Feedback Descent	Summary	pairwise comparison + feedback	0.012
TTT-Discover	Window	prev. solution fragment	0.026
Meta-Harness	Full	all logs and scores	10.0

Context available per optimization step. Mtok/iter = estimated context per evaluation in each paper's largest setting. Hover a row for details.

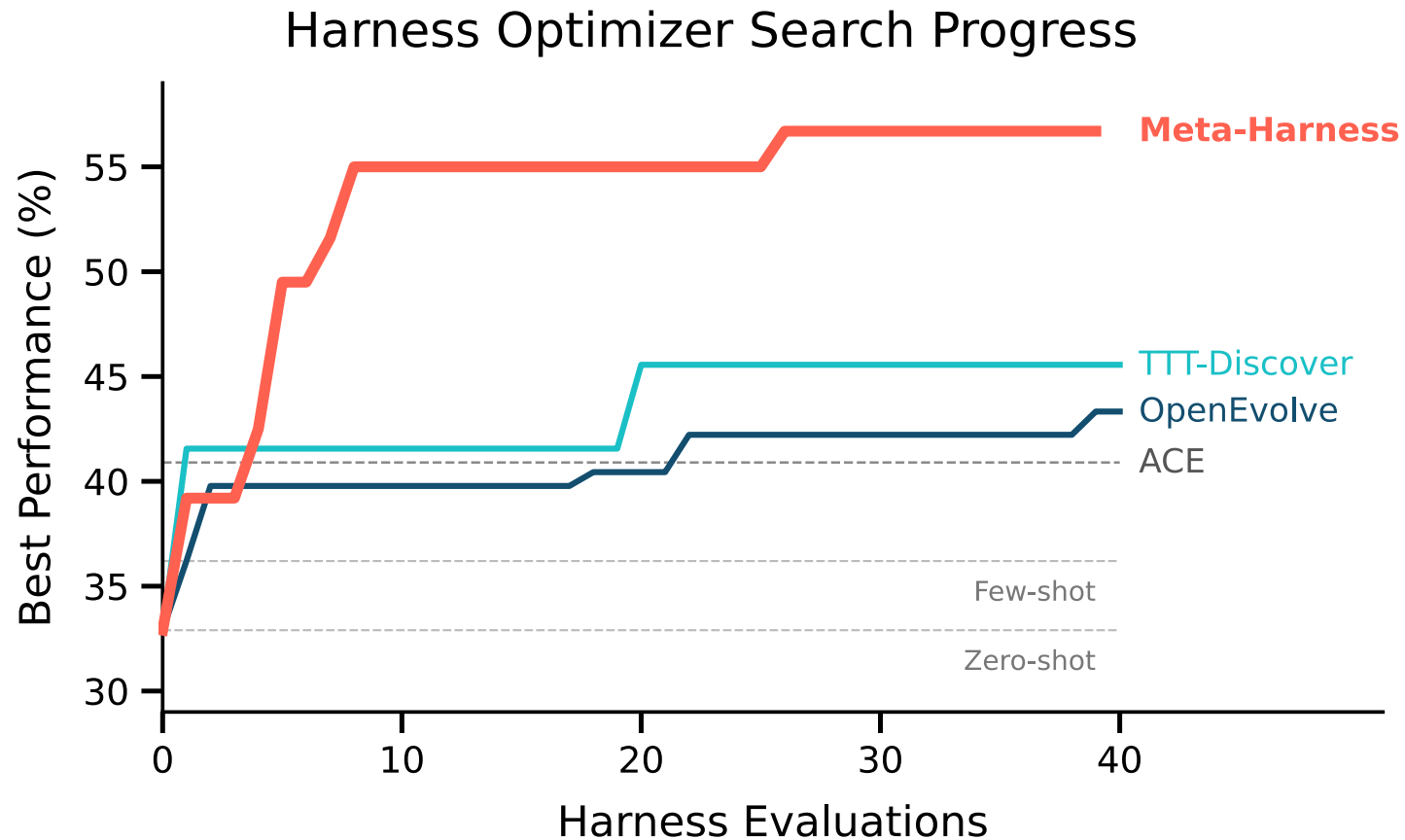
Meta-Harness

A unifying framework represented as file systems, fully optimized through code.
Optimizes prompts, RAG systems, skills, tools, memory, full program loops.



Meta-Harness

Use case 1: Evolving a harness for text classification



Meta-Harness

Use case 2: Evolving RAG harness for scientific research, context used for math reasoning

Method	GPT-5.4n ↑	GPT-5.4m ↑	Gem-3.1FL ↑	Gem-3F ↑	GPT-20B ↑	Avg ↑
No Retriever	23.0	28.8	28.6	42.6	47.6	34.1
Random Few-shot	23.1 (+0.1)	24.5 (-4.3)	31.0 (+2.4)	40.4 (-2.2)	41.8 (-5.8)	32.2 (-1.9)
BM25 Retrieval	30.2 (+7.2)	29.2 (+0.4)	32.8 (+4.2)	46.6 (+4.0)	48.9 (+1.3)	37.5 (+3.4)
Meta-Harness (Ours)	31.7 (+8.7)	30.4 (+1.6)	34.9 (+6.3)	46.3 (+3.7)	50.6 (+3.0)	38.8 (+4.7)

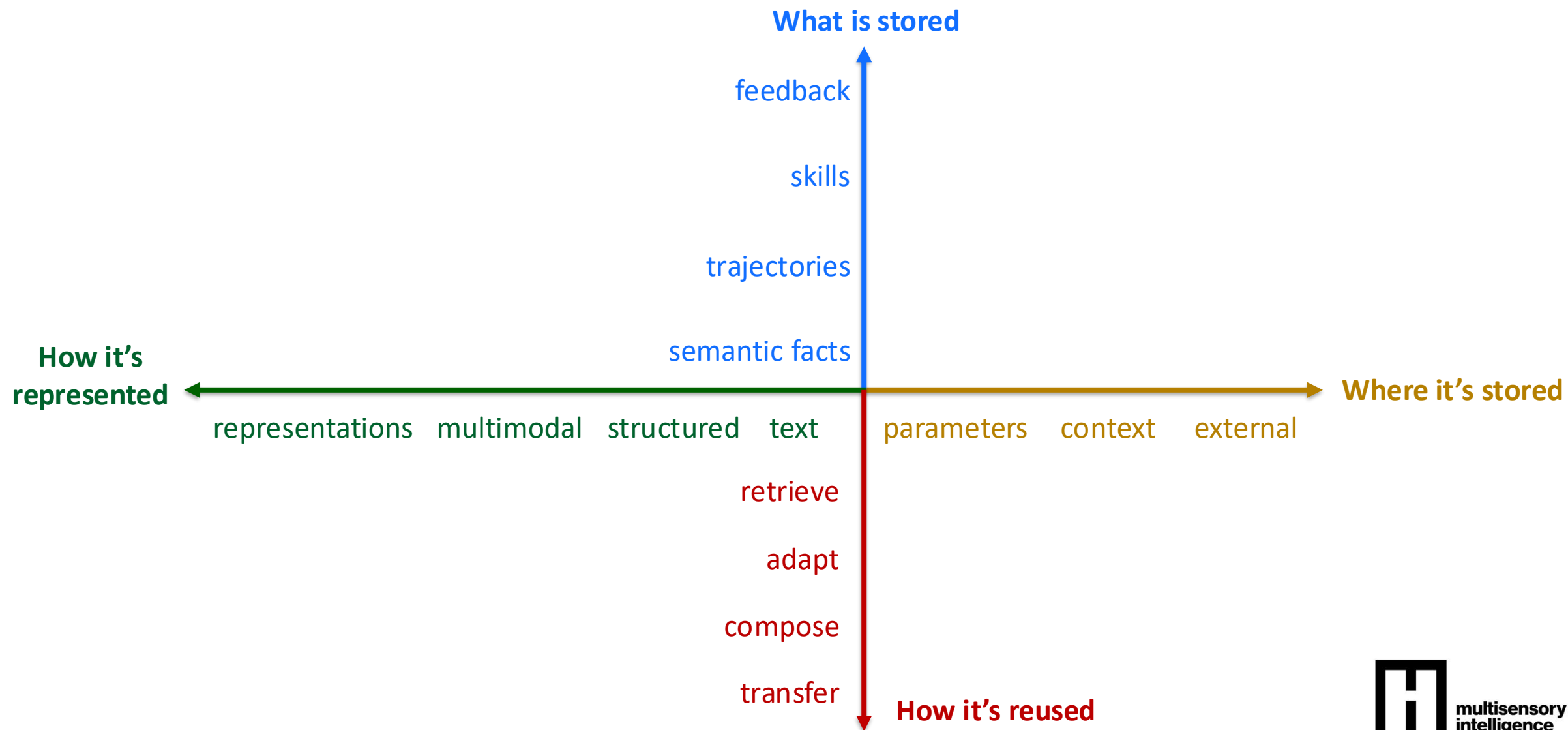
Meta-Harness

Use case 3: Agentic coding with long-horizon autonomous execution + verification loop

Claude Opus 4.6	Pass %
Claude Code	58.0
Terminus 2	62.9
Mux	66.5
Factory Droid	69.9
TongAgents	71.9
MAYA-V2	72.1
Terminus-KIRA	74.7
Capy	75.3
Meta-Harness (Ours)	76.4

Claude Haiku 4.5	Pass %
OpenHands	13.9
Claude Code	27.5
Terminus 2	28.3
Mini-SWE-Agent	29.8
Terminus-KIRA	33.7
Goose	35.5
Meta-Harness (Ours)	37.6

Summary of Harnesses



Summary of Harnesses

Eg. In a coding agent harness

Episode/trajectory: state, action, next state, next action (e.g., good debugging examples)

Feedback: rewards, critiques, human preferences (e.g., user prioritizes clean code)

Facts: semantic knowledge (e.g., sorting function has certain complexity)

Tools: task-specific atomic (external) action (e.g., running python interpreter)

Skills: task-agnostic reusable procedure (e.g., use gdb to debug)

Artifacts: generated or optimized outputs (e.g., generated code)

Plans: decomposed tasks and sub-goals (e.g., read requirements, write, modularize)

Experience: often used vaguely, encompasses trajectories, feedback, skills

Summary of Harnesses

My favorite methods out of the ones covered today:

MEM1: treat memory as internal state in context autonomously updated by LLM

CLM: treat entire context as persistent file autonomously maintained and rewritten by LLM

Meta-Harness: treat entire harness as a file system autonomously modified by LLM

What's common about all these methods?

- Simple, no hand engineering, end-to-end
- New paradigm that generalizes what can be autonomously optimized (memory -> context -> harness)
- Scalable results for zero-shot, in-context prompting, fine-tuning