Memory and Skills

How agents improve without starting over

Carnegie Mellon University
Language Technologies Institute

Daniel Fried
11-768: AI Agents

Shared task structure

	SHARED SEARCH PROCEDURE				TASK-SPECIFIC FINISH		
TASK 1 **Add Sony headphones to my wish list**	`goto store`	→ `find search box`	→ `fill("sony headphones")`	→ `click Search`	`open Sony result`	→ `Add to Wish List`	
TASK 2 **Find the wireless keyboard price range**	`goto store`	→ `find search box`	→ `fill("wireless keyboards")`	→ `click Search`	`read result prices`	→ `report min–max`	

Lots of structure recurs in agent tasks.

TODAY'S QUESTIONS:

REPRESENTATION

Should the shared structure be text or code?

INDUCTION

How can the agent induce it from experience?

LIFECYCLE

When should a skill be retrieved, checked, revised, or forgotten?

Methods of updating the agent

LOCATION	PROS	CONS
Context window LAST LECTURE	high fidelity to what happened	costly, noisy, does not decide what matters
External artifacts THIS LECTURE	inspectable, editable, retrievable, portable	must be induced, selected, and maintained
Model weights SFT LECTURE	faster inference, broad behavior change	slower updates, opaque, model-specific

What experience is worth turning into an external artifact?

Types of experience

Episode

A full trace, every observation and action

Useful if the exact product, prices, or sequence will matter again.

Fact

“This store shows prices only on the product page.”

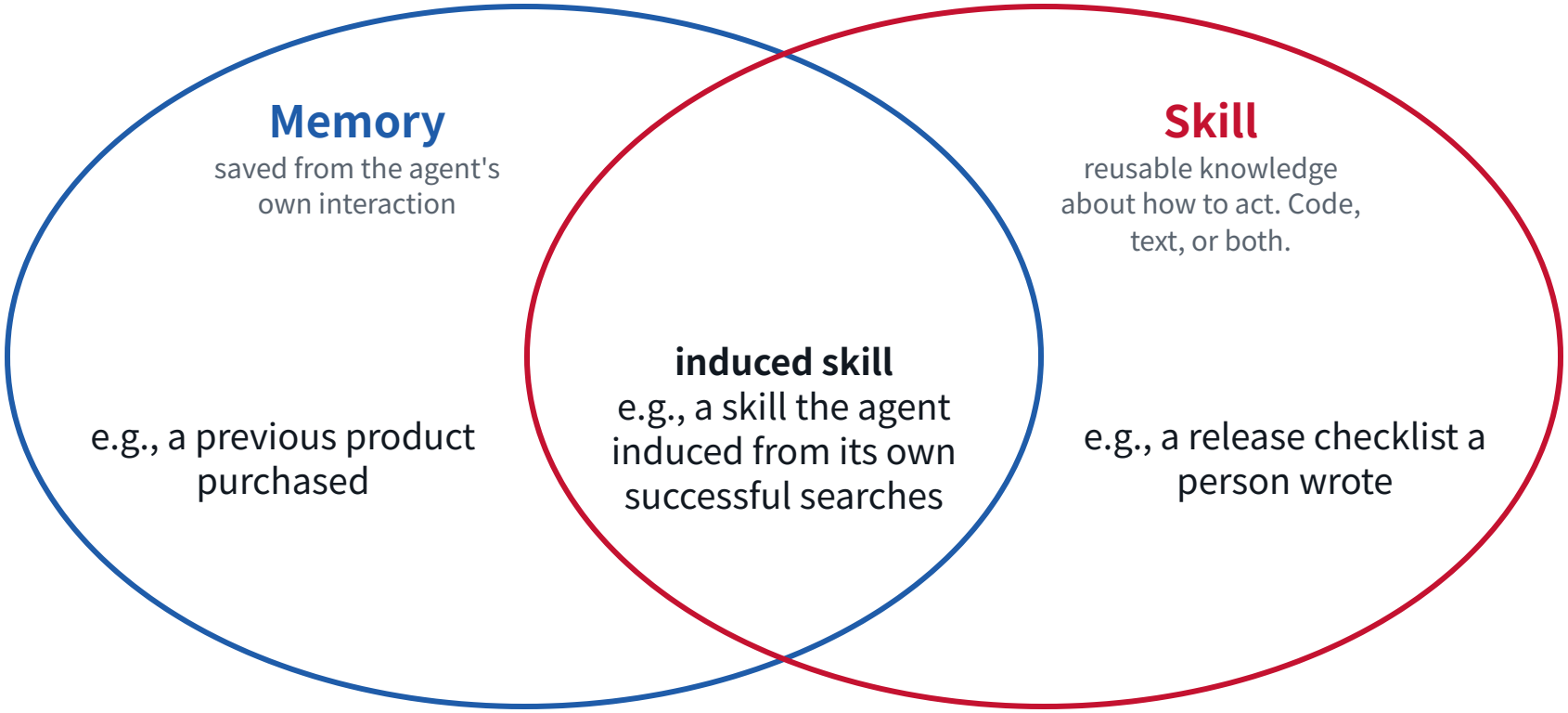
Useful if the fact recurs in future tasks, or can be updated.

Skill

fill the search box → click search → open a result

Useful if later tasks will repeat the same pattern.

Memory versus skill



Skill creation pathways

	SKILLS AS INSTRUCTIONS	SKILLS AS LEARNING
Author	a person or organization	the agent
Source	existing expertise, policy, documentation	its own episodes and outcomes
Strength	known intent and provenance	works where the skill must be discovered
Main burden	expert time to write and maintain	judging outcomes, verification, curation

Authored skills

What is a skill?

THE SAME REQUEST, TYPED INTO EVERY PULL-REQUEST CONVERSATION

```
Please check this Python code using:
- Black formatting with 88-char line limit
- Ruff linting with our custom rules
- Type hints for all public APIs
- Google-style docstrings
- Pytest for all new functions
```

- **When you want to do something repeatedly following a fixed spec.** e.g. adding tests, reviewing pull requests, upgrading dependencies
- **Here a person wrote it.** Later in the lecture the agent is asked to notice the repetition and write the skill itself
- **Implemented using a collection of files.** Documentation, scripts, and resources that provide guidance and automation

.

OpenHands · Michelini & Neubig, “How to Create Effective Agent Skills”, 2026 ↗

Example of an authored skill

PACKAGE

```
python-review/
├── SKILL.md          # Required: instructions + metadata
├── scripts/          # Optional: executable code
├── references/       # Optional: documentation
└── assets/           # Optional: templates, resources
```

SKILL.MD

```yaml
---
name: python-review
description: This skill should be used when the user asks to
  "review Python code", "check Python style", "lint Python",
  or mentions code quality.
triggers:
  - python review
  - code review
  - lint python
---
# Python Code Review
Review Python code using company standards ...
```

LEVEL	FILE	CONTEXT WINDOW
1	`SKILL.md` metadata (YAML)	**Always loaded**
2	`SKILL.md` body (Markdown)	**Loaded when the agent triggers the skill**
3	Bundled files (text, scripts, data)	**Loaded when the agent reads the files**

Progressive disclosure: Names and descriptions load up front; the rest is read only when the agent calls for it.

Example: OpenHands, “How to Create Effective Agent Skills” ↗ Table: Anthropic, “Equipping agents for the real world with Agent Skills” ↗ AgentSkills specification ↗

Progressive Disclosure: Indexing skills

SYSTEM PROMPT

```
## Skills
Before replying, scan the skills below. If a skill matches or is even
partially relevant to your task, you MUST load it with skill_view(name)
and follow its instructions. …

<available_skills>
  code-quality:
    - python-review: This skill should be used when the user asks to
      "review Python code", "check Python style", "lint Python", or
      mentions code quality.
  workflow:
    - submit-task: When the task is complete, take the required steps to
      submit the solution. DO NOT submit without invoking this skill.
    - release-notes: …
</available_skills>

Only proceed without loading a skill if genuinely none are relevant.
```

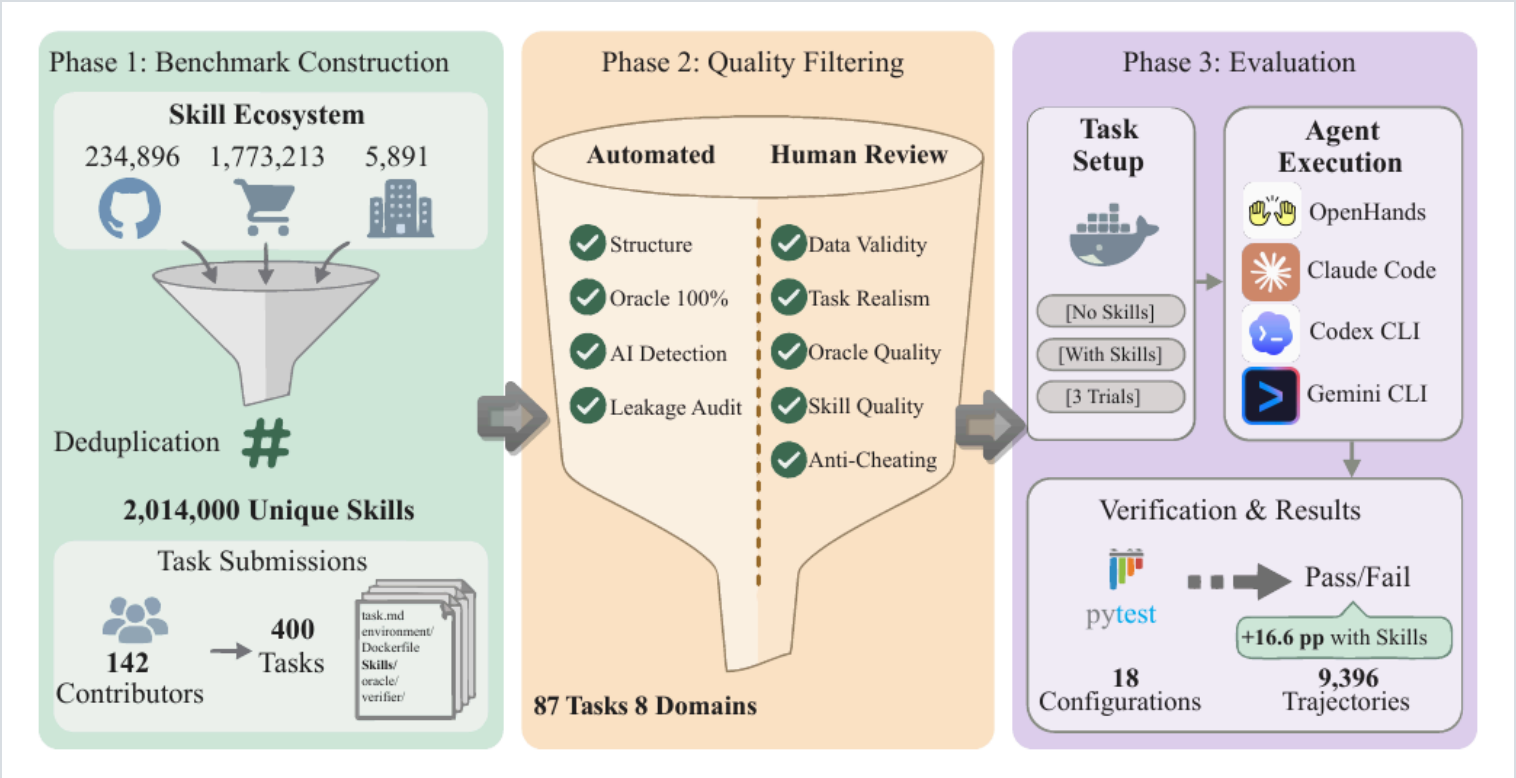
Abridged from HermesAgent · agent/prompt_builder.py (MIT) ↗

Progressive Disclosure: loading skills with tools

TURN	WHAT ENTERS THE CONTEXT
1 System Prompt	the index of skills: names and descriptions only *≈3k tokens in Hermes*
2 `skill_view('python-review')`	the `SKILL.md` body, as a **tool result**: the review steps, as text *once;* `invoke_skill` *in Assignment 1*
3 **TERMINAL** `python scripts/run_checks.py src/`	the script's **output**: Black and Ruff findings *the code itself never enters*
4 `skill_view('python-review', 'references/docstring-style.md')`	the docstring rules *only when the model calls for them*

HermesAgent · tools/skills_tool.py ↗ Assignment 1 · § 1.4 Load skills ↗ Anthropic, Agent Skills ↗

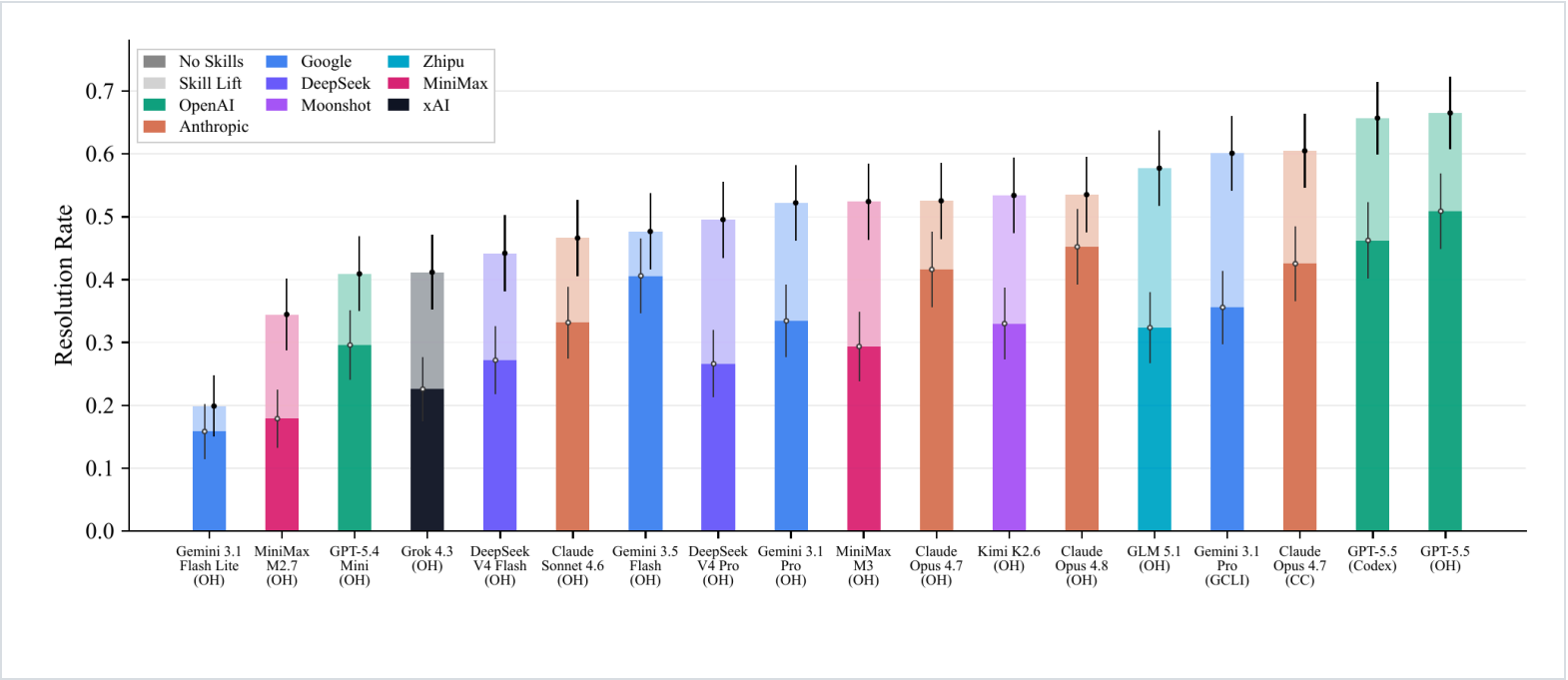
Measuring improvements from skills



- **SkillsBench: 87 real-work tasks across 8 domains.**
- **Curated task–skill pairs.** Humans choose the skill bundle supplied with each task
- **Main experiment: no skills vs. curated skills.** 18 model–harness configurations × 87 tasks.

Figure: SkillsBench · arXiv 2602.12670 ↗

Gains and limits of human-authored skills



Skills raise the average pass rate from 33.9% to 50.5%.

- **Focused skills correlate with larger gains.** Tasks with 1–3 skills improve more than tasks ≥4 skills; shorter SKILL.md groups also show larger gains.
- **Skills still *hurt* on 13 of 87 tasks.** A skill can displace a better default strategy.

SkillsBench · arXiv 2602.12670 ↗

Sources of skill packages

Bundled

Ships with the harness as part of its default set.

Installed or written

A person writes one for their own work, checks it into the repository under `.agents/skills/`, or installs it from another repository or a registry.

Created by the agent

The agent writes the same package itself. Hermes exposes a `skill_manage` tool that can create, patch, or delete; OpenHands has a `/skill-creator` command that drafts one from the workflow the agent just finished. A person can review either before it ships.

OpenHands' advice: don't write a skill from scratch. Create it right after the agent finishes a workflow you want to repeat, then test it a few times.

HermesAgent skills documentation ↗ OpenHands, “How to Create Effective Agent Skills” ↗

Case study: maintaining a code-review skill

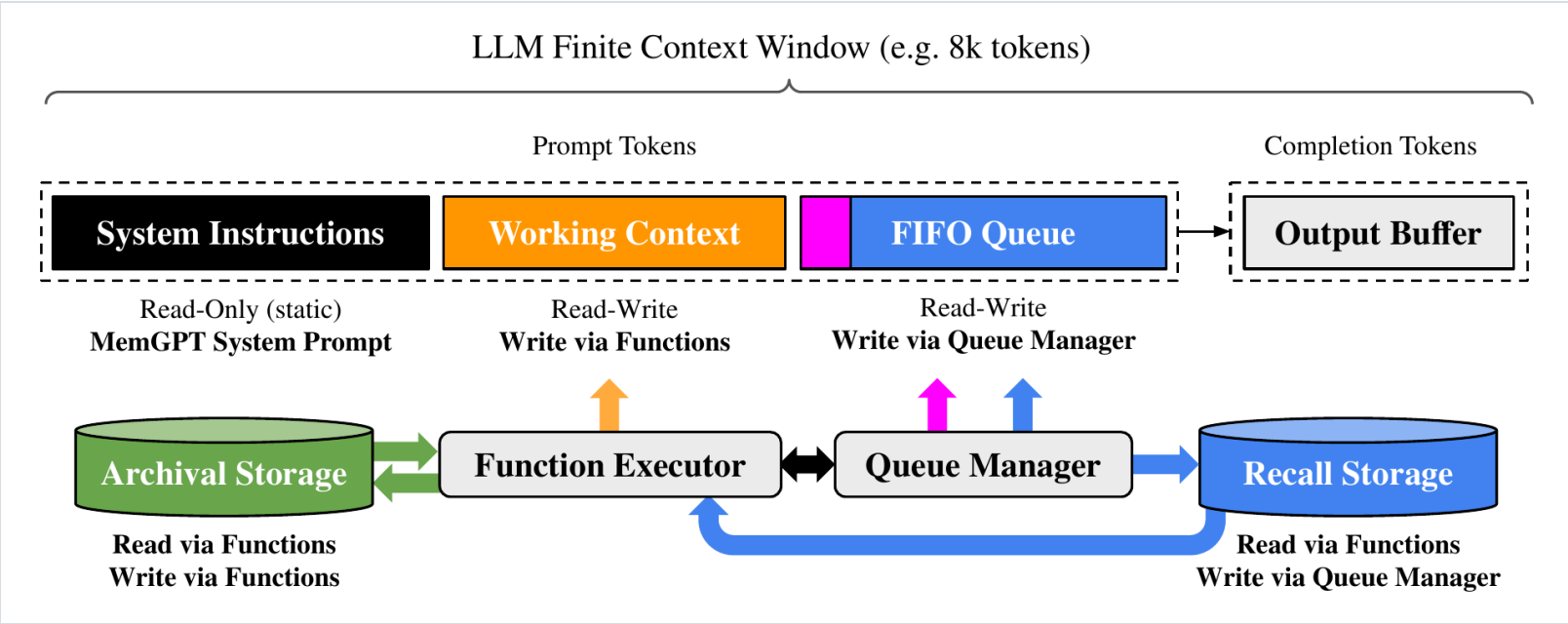
OpenHands runs a review skill of the `python-review` kind on every pull request in its own repos.

- **Evaluate the outputs.** After each PR merges, a model judge counts how many of the agent's suggestions the humans kept.
- **Have a model analyze failures.** It reads the run notes and names what recurs: ignoring repo conventions (~15%) and approving PRs with a critical flaw (~10%).
- **Update `SKILL.md`.** The same model drafts it: request changes whenever a critical issue is found, and check a suggestion against the repo before posting it.

OpenHands · Michelini & Neubig, “How to Create Effective Agent Skills”, 2026 ↗

Memories: facts, lessons, and episodes

Managing an external memory



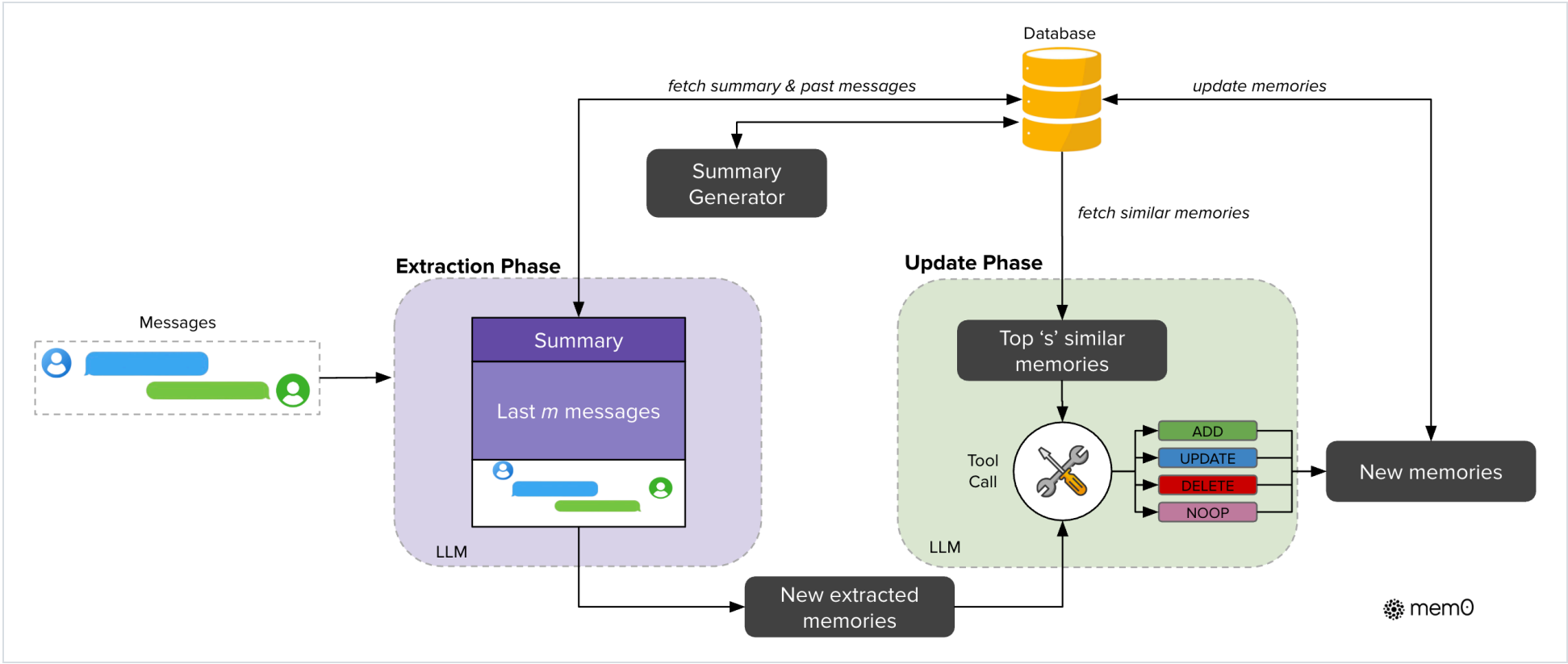

```
recall_storage.search("six flags")
```

Showing 3 of 3 results (page 1/1):
[01/24/2024] "lol yeah six flags",
[01/14/2024] "i love six flags been like 100 times",
[10/12/2023] "james and I actually first met at six flags"

Reads and writes are function calls the agent issues itself.

Figures 3 and 2: MemGPT · Packer et al., 2023 ↗

Handling memory conflicts



ADD

nothing similar is stored yet

UPDATE

messages change an existing memory

DELETE

messages negate an existing memory

NOOP

messages are already stored in memory, or irrelevant

Figure: Mem0 · Chhikara et al., 2025 ↗

Remembering feedback on trajectories

attempt → **evaluator: predict reward** → **reflection: generate feedback** → retry the same task

ATTEMPT 1

"What was a series of battles … fought on October 28, 1776 near White Plains, New York?"

`Finish[Battle of White Plains]` **incorrect**

GENERATED FEEDBACK

"… the question asked for a series of battles, but I only provided the name of one battle … I will make sure to provide more context, such as the name of the campaign …"

ATTEMPT 2

`Finish[The New York and New Jersey campaign]` **correct**

Reflexion · Shinn et al., NeurIPS 2023 ↗

Similar-trajectory conditioning

- **What is stored:** whole trajectories, both successes and failures, sometimes rewritten and annotated before storage.
- **Where the pool comes from:** collected offline in a training phase, grown while the agent works, and/or hand-written in advance.
- **How it is used:** retrieve the runs most similar to the new task using text embeddings, and include them as few-shot examples.

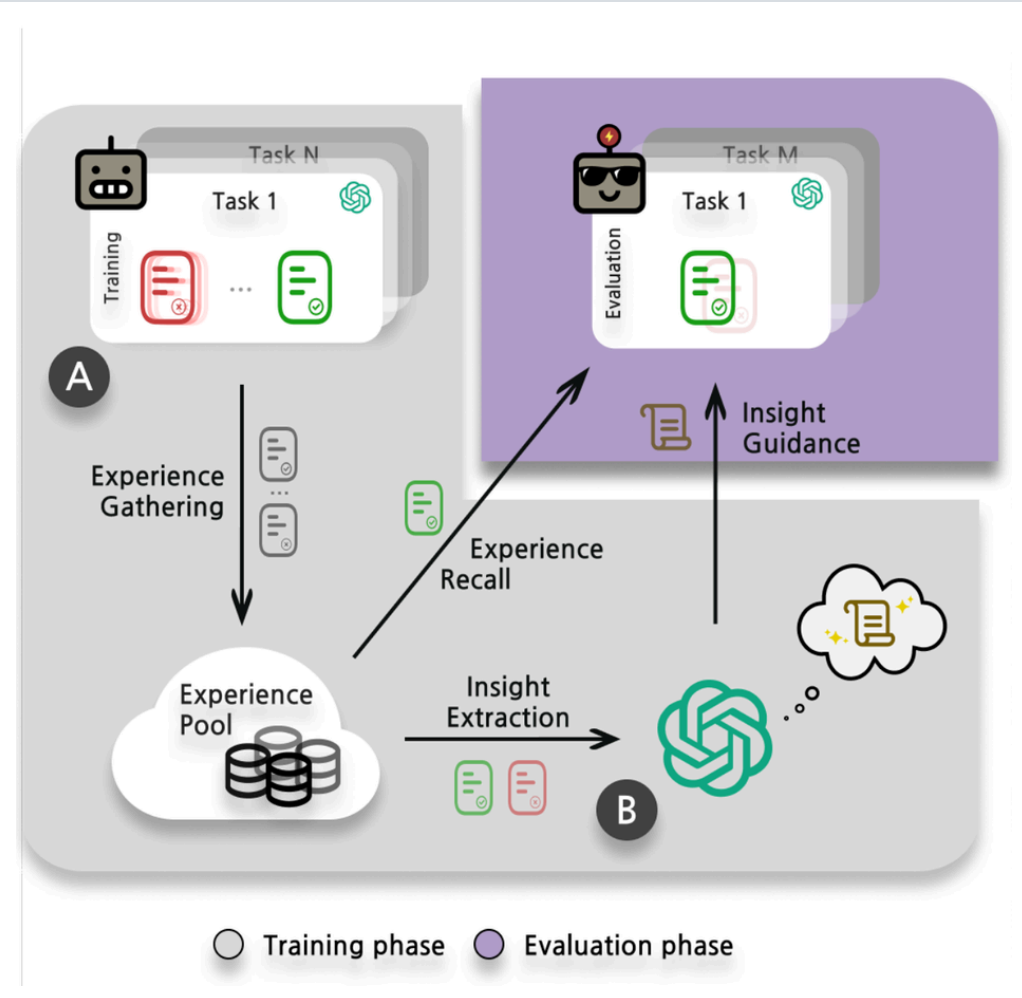


Figure from ExpeL, Zhao et al. 2024

ExpeL · Zhao et al., AAAI 2024 ↗ Agent S · Agashe et al., ICLR 2025 ↗ Synapse · Zheng et al., ICLR 2024 ↗ ICAL · Sarch et al., NeurIPS 2024 ↗

Inducing skills

Skills as reusable chunks of a task

SEARCH FOR A PRODUCT

Show me results for {query}

To find {query}, I will type it into the store's search box and submit.

```
fill('element145', {query})
click('element147')
```

Reuse →

Add steps ↘

REPORT A PRICE RANGE

What is the price range for {query}?

To get the price range for {query}, I will first search the store for it.

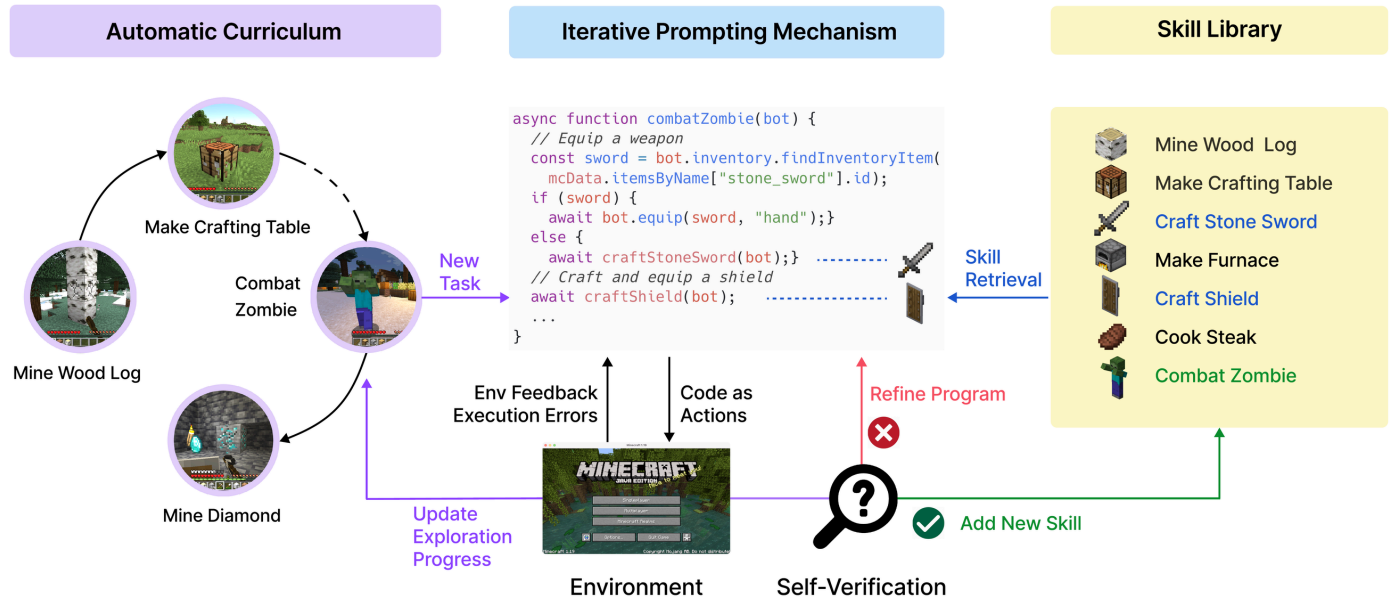
```
fill('element145', {query})
click('element147')
```

The results list a price on each card. I will read the lowest and the highest.

```
send_msg_to_user("{low} to {high}")
```

How should a skill be represented? We'll compare text vs code.

Skills / program induction in past work



- **Voyager.** Induce functions for increasingly complex action sequences in Minecraft, each calling earlier ones.
- **DreamCoder.** Induce functions that compress a corpus of solved programs; later search reuses them.
- **Stitch.** Fast symbolic compression: find functions that capture the most shared structure in a corpus.
- **LAPS.** Natural-language task descriptions guide which functions are induced and how programs are searched for.
- **LILO.** An LLM writes programs; Stitch compresses them into functions; the LLM names and documents each.

Figure: Voyager · Wang et al., 2023 ↗ DreamCoder · Ellis et al., 2021 ↗ Stitch · Bowers et al., 2023 ↗ LAPS · Wong et al., 2021 ↗ LILO · Grand et al., 2024 ↗

Inducing skills online

EVALUATION SETTING

A successful evaluation task updates memory before the next task arrives.

Evaluate cumulative success rate over all tasks seen so far

SKILL MEMORY

search for a product

add a product to the wish list

report a price range

grows across the evaluation stream

① induce

after success ←

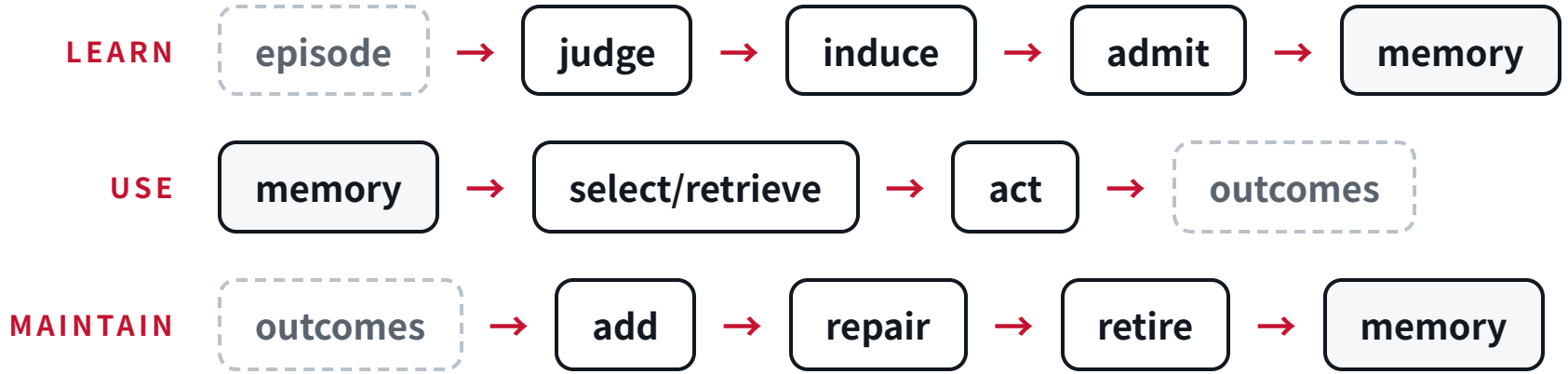
→ on a later task

② apply

EVALUATION TASKS ARRIVE SEQUENTIALLY

1. add a Sony bluetooth headphone to my wish list
2. find gaming accessories added in the last month
3. what is the price range for wireless keyboards?

The lifecycle of an induced skill



Inducing textual representations of skills

Zora Wang

LEARN judge → induce → admit

episode

NL query

Tell me the number of reviews that our store received so far that mention the term “satisfied”

\+ Actions

Memory

```
click(“Marketing”)
click(“All Reviews”)
fill(757, “satisfied”)
click(“Search”)
send_msg_to_user(“2”)
```

Judge

Final state predicted to be correct?

YES ▸ NO ▾ *pass*

Few-Shot Prompting

LM

Induce

Task: Search for reviews matching “{term}”

Actions:

```
# To find reviews for {term}, I will search for “{term}” in the reviews page
click(“All reviews”)
fill(757, “{term}”)
click(“Search”)
```

Memory

Agent Workflow Memory · Wang et al., ICLR 2025 ↗

An induction prompt

LEARN judge → induce → admit

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.
```

AWM · Wang et al., ICLR 2025 — Appendix A.1 ↗

Cross-task memory compounding

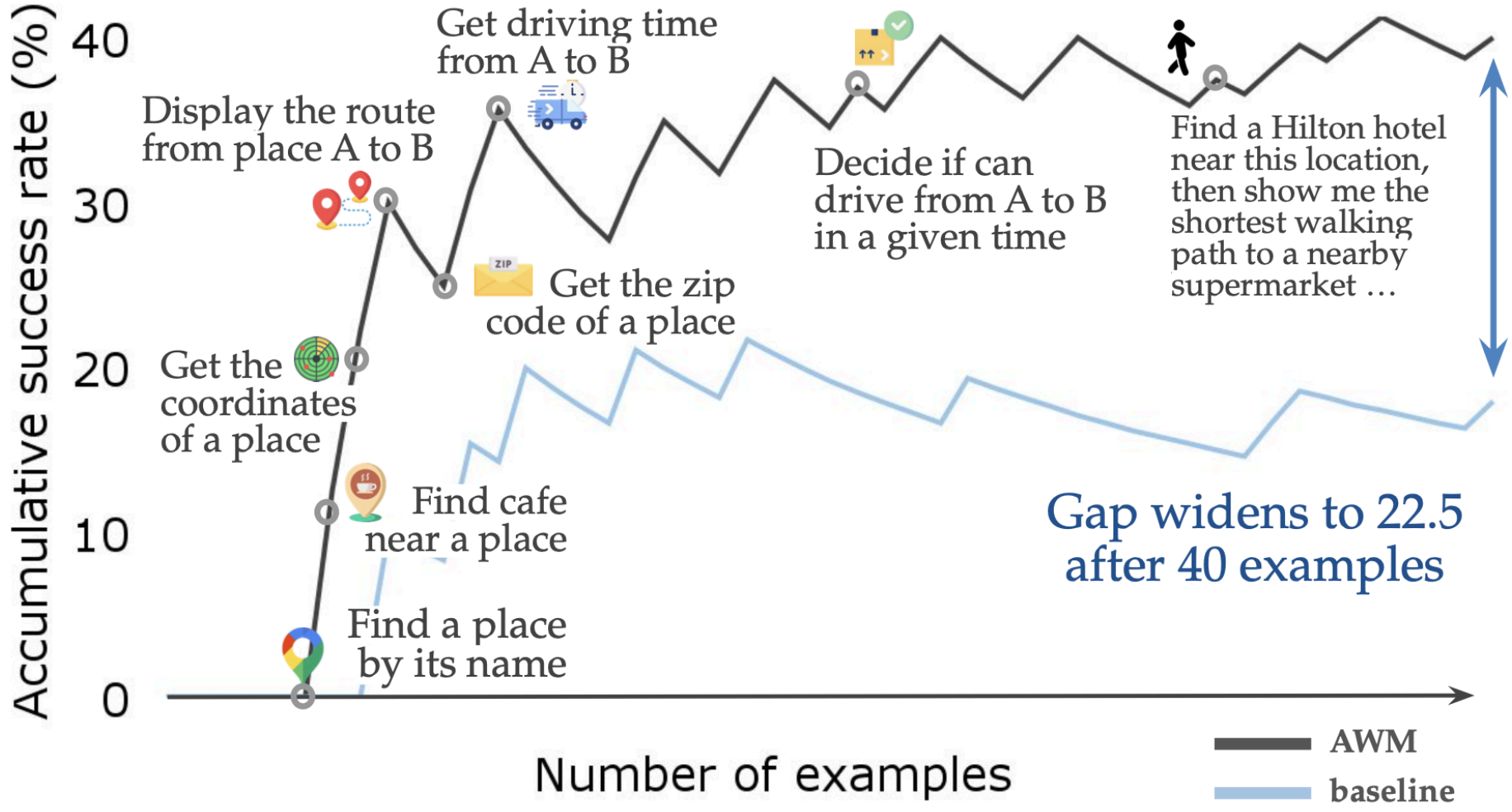


Figure: AWM · Wang et al., ICLR 2025 ↗

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

```python
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

Workflows · AWM, Wang et al., ICLR 2025 ↗ Skill · ASI, Wang et al., COLM 2025 ↗

Inducing code skills

Zora Wang

LEARN judge → induce → admit

episode

NL query

Tell me the number of reviews that our store received so far that mention the term “satisfied”

\+ Actions

```
click(“Marketing”)
click(“All Reviews”)
fill(757, “satisfied”)
click(“Search”)
send_msg_to_user(“2”)
```

Few-Shot Prompting

LM

Induce

skills

```
def search_reviews(search_box_id, search_button_id, search_term):
  """Search for reviews with a specific term.
  Args: search_box_id: … …
  Examples: search_reviews('757', '704', 'great')"""
  fill(search_box_id, search_term)
  click(search_button_id)

def open_marketing_reviews():
  """Navigate to All Reviews under Marketing.
  Examples: open_marketing_reviews()"""
  click("Marketing")
  click("All Reviews")
```

test

```
open_marketing_reviews()
search_reviews('757', '704', "satisfied")
~~send_msg_to_user("2")~~
```

Judge

Final state predicted to be correct? Skills used?

YES ▸ **NO** ▾ *pass*

Memory

After ASI · Wang et al., COLM 2025 ↗

Lineage: Voyager grew a memory of code skills in Minecraft ↗

Code skills allow testing

LEARN judge → induce → admit

CODE SKILL ADMISSION

```
candidates = propose_skill(task)
episode = practice(candidates, task)

if reward_model(episode.actions, episode.screenshots, task):
    memory.add(candidate)
else:
    candidate = revise(candidate, task)
```

EXAMPLE OF LINTING AND TESTING A CODE SKILL

```
identify_pill(page, imprint="M366", color="White")
```

```
WARNING  parameter 'color' is defined but never used in the function body
```

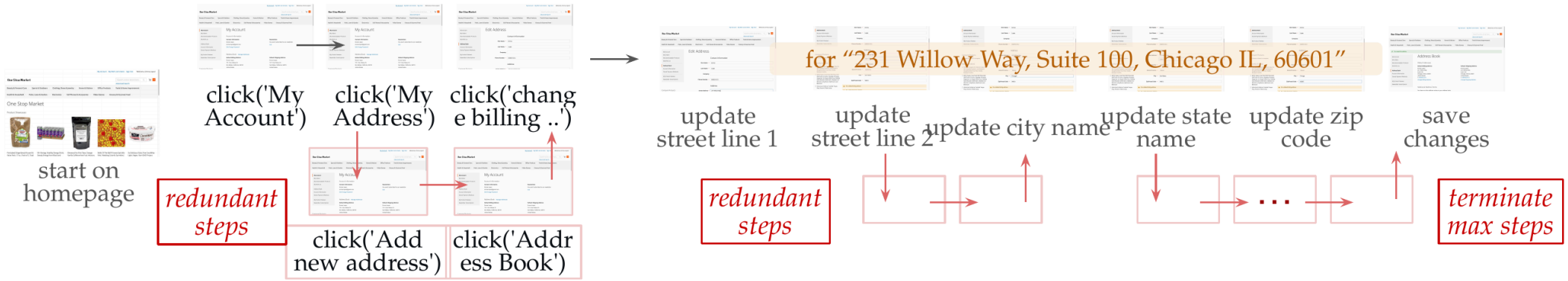
```
+ if color:
+     await page.get_by_role("group", name="Color and shape (optional)") \
+                 .get_by_role("combobox", name="Color (optional)").select_option(color)
```

After SkillWeaver · Zheng et al., 2025 — Figure 1 ↗

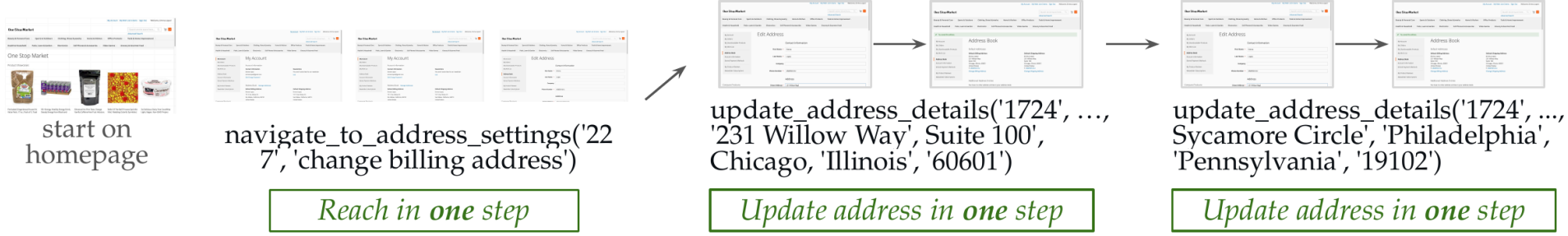
Task compression with code skills

I recently moved. Can you change my billing address to "231 Willow Way, Suite 100, Chicago, IL, 60601"? Then, update my shipping address to: 987 Sycamore Circle, Philadelphia, PA, 19102.

no memory
50 steps, cap

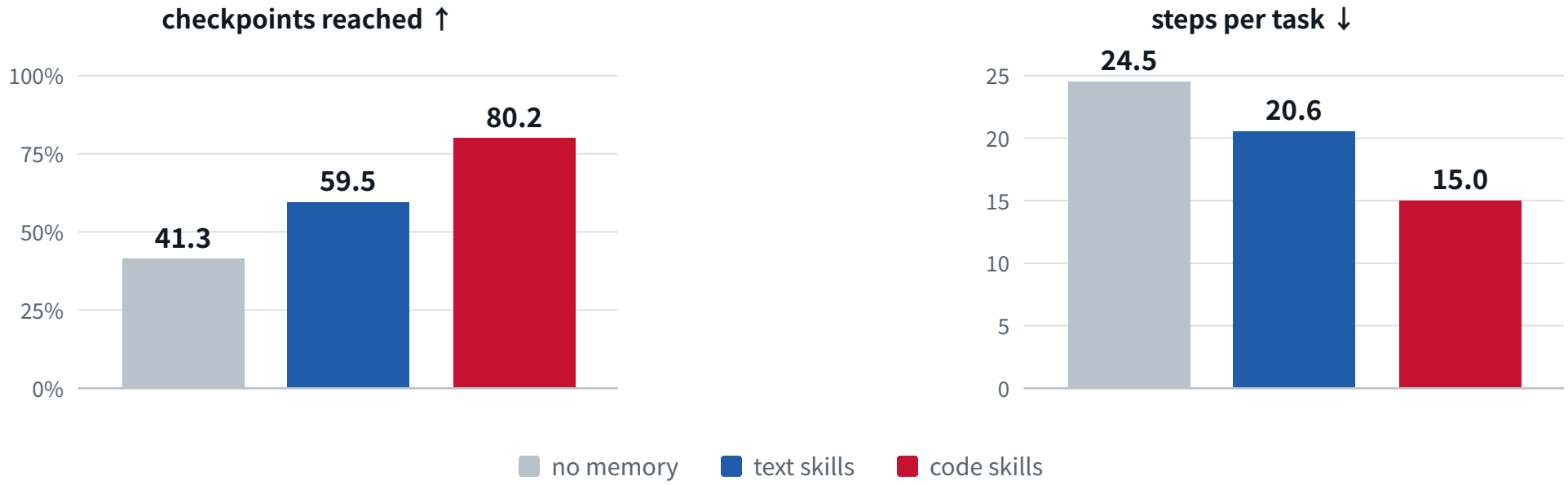


code skills
4 steps



ASI · Wang et al., COLM 2025 — Figure 5, shopping site ↗

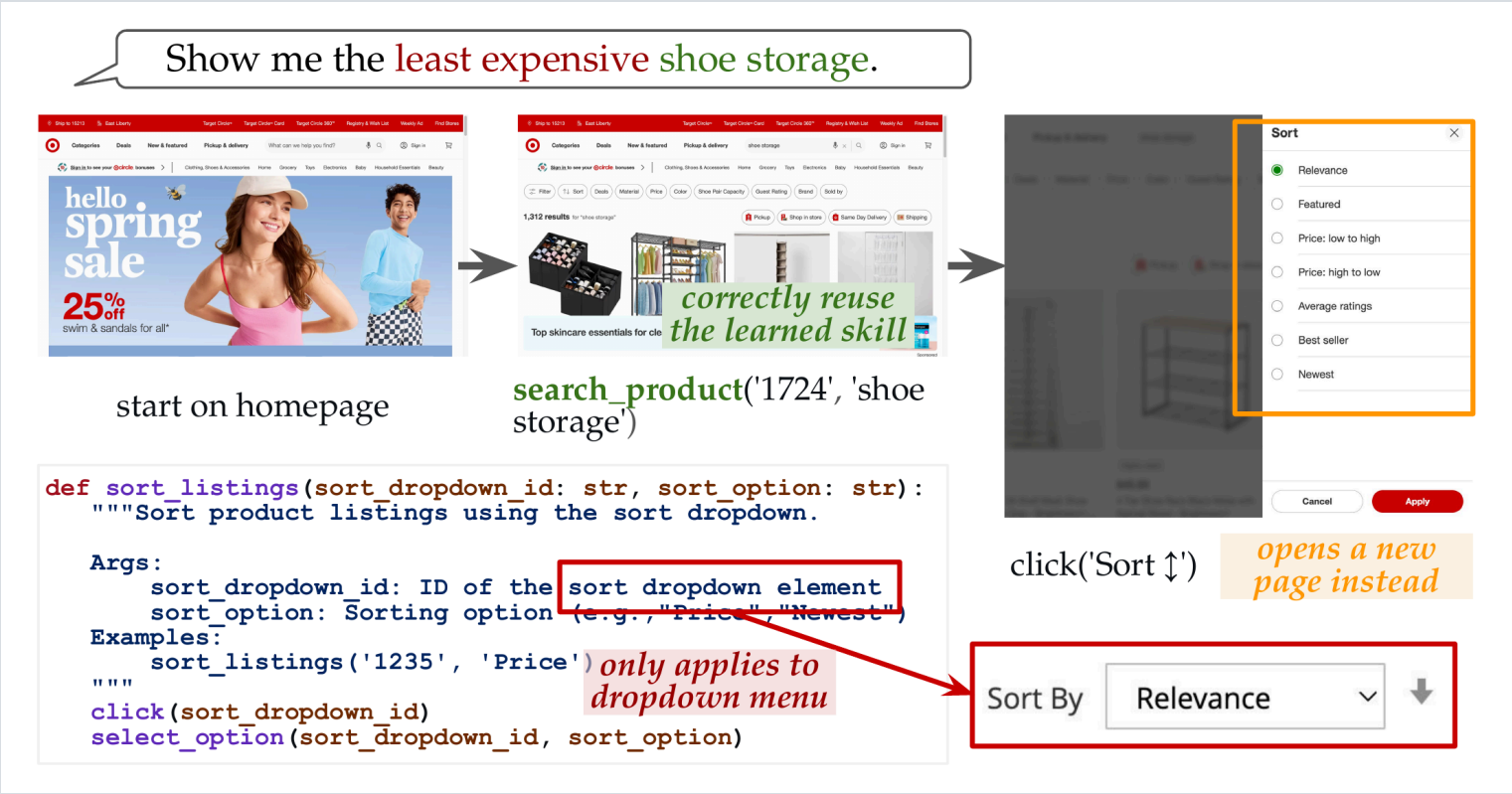
Code skills raise success and cut steps



- Comparison between text and code skills on web tasks with repeated structure.
- Code skills generally improve success (checkpoints reached) over text skills.
- They also use fewer steps by the agent.

ASI · Wang et al., COLM 2025 — Table 2, averaged over five WebArena sites ↗

How general are skills?



- Induce skills on tasks on one website; evaluate on another.
- `sort_listings` clicks a dropdown; Target's sort opens a sidebar.

ASI · Wang et al., COLM 2025 — Figure 4, WebArena skills on target.com ↗

PolySkill: one interface, many implementations

MAINTAIN repair → retire

PolySkill Abstract Class

```python
# 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

```python
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)
```

```python
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)
```

Compositions are written once against the abstract methods; each website implements only the primitive browser actions.

PolySkill · Yu et al., ICLR 2026 — Table 1 ↗

Consequences of skill representation

REPRESENTATION	STRENGTHS	WEAKNESSES
Retrieved episode	preserves concrete behavior	long, hard to transfer
Text skill	provides flexible guidance	doesn’t improve efficiency
Code skill	executable, composable, efficient	can be brittle

How much of the behavior is left to the model, and how much is fixed in the artifact?

The skill lifecycle

Learning from failures

LEARN judge → induce → admit

EPISODE

“Provide me with the complete names of Bluetooth headphones from Sony, and also share the price range for the available models.”

`search “Bluetooth headphones Sony” → 5,578 results from every department of the store, 12 per page → next page → next page → … → step limit, nothing sent to the user`

Failure.

INDUCED STRATEGY

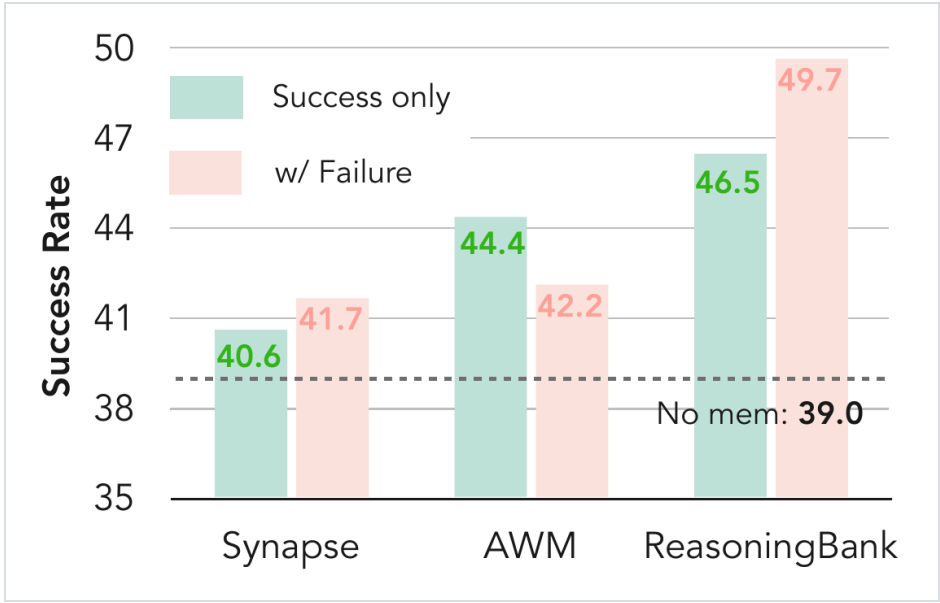
“Search query optimization” — a vague query buried the answer under thousands of loosely matched results. Tighten the query before browsing.

“Adjust number of items displayed per page” — twelve per page makes every “next” cost a step for little coverage.

“Use filters available” — the sidebar’s category filters shrink the list without paging.

ReasoningBank · Ouyang et al., ICLR 2026 — appendix case study ↗

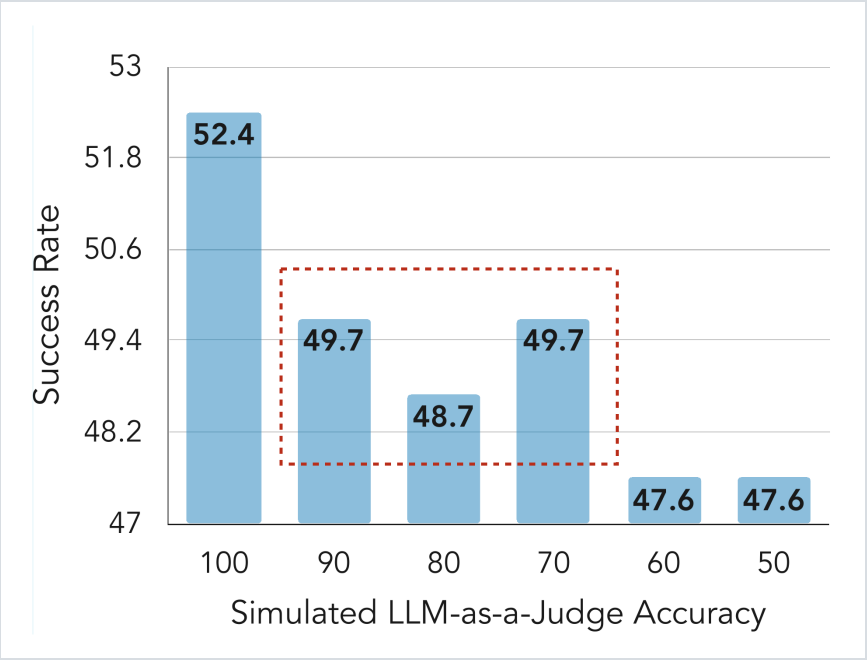
Failures help some memory types and hurt others



- Failures help substantially when inducing strategies (ReasoningBank)
- Failures are less helpful with trajectories (Synapse) or workflows (AWM): demonstrations of what not to do?

ReasoningBank · Ouyang et al., ICLR 2026 — WebArena shopping, Gemini-2.5-flash ↗

How much does judge accuracy matter?

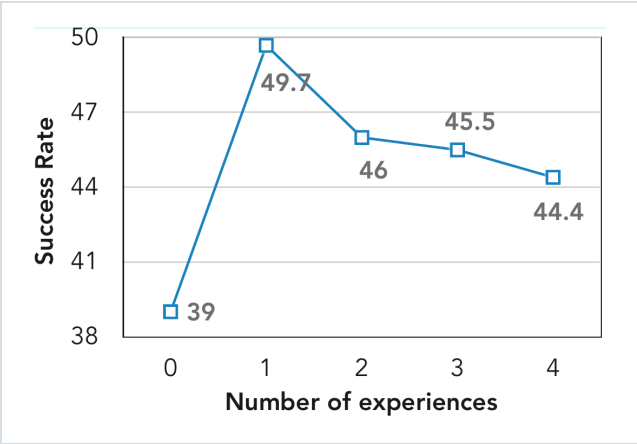


- **Setup:** the real judge is imperfect. To ask how much that matters, they swap in simulated judges: the true label, flipped with a fixed probability, from oracle to coin flip.
- **Result:** performance depends on the judge accuracy, but performance improves for a wide range of judge qualities.

ReasoningBank · Ouyang et al., ICLR 2026 — judge calibration analysis ↗

The cost of over-retrieval

USE **retrieve** → act

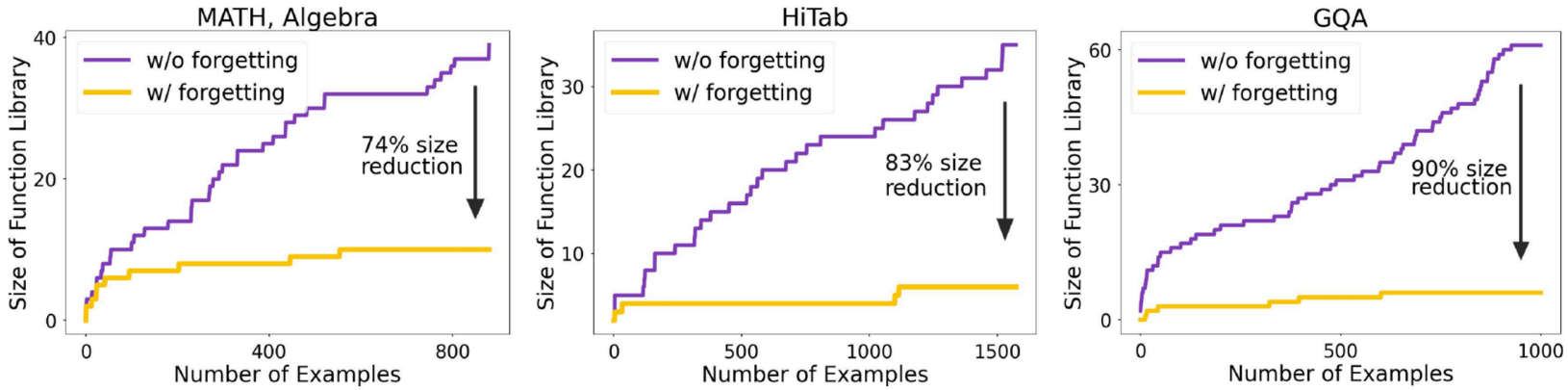


- Retrieving additional experiences can cause performance to degrade!
- Example above from ReasoningBank.
- SkillsBench found the same for human-curated skills.
- Some experiences may be irrelevant: limitations of the memory
- The agent may be limited in what it can condition on

ReasoningBank · Ouyang et al., ICLR 2026 — WebArena shopping ↗

Memory consolidation

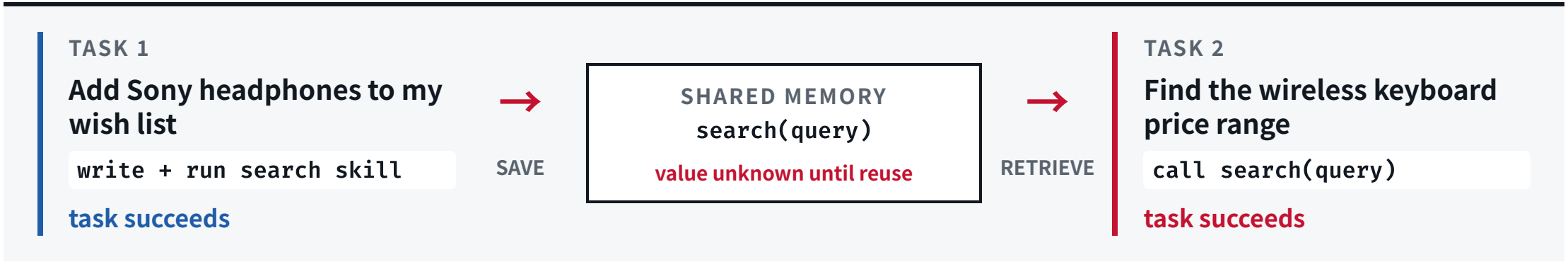
MAINTAIN **repair** → **retire**



- TroVE drops any function used fewer than $\frac{1}{2} \cdot \log_{10}(n)$ times, where n is the number of examples seen so far.

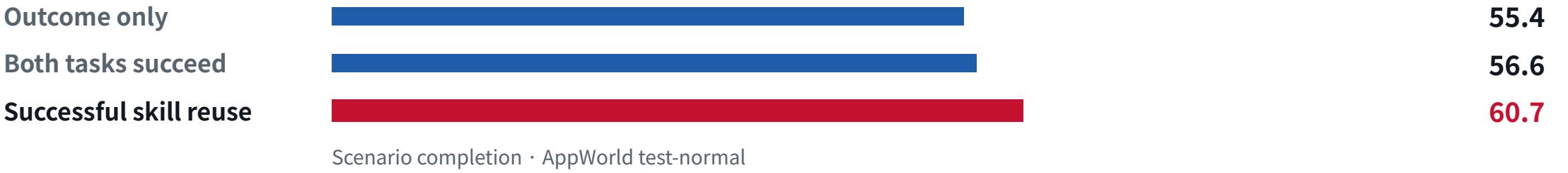
TroVE · Wang et al., ICML 2024 ↗ Also: Not All Skills Help — a skill that helps one task type hurts another ↗

Training the skill inducer



Task 1's reward says whether the task succeeded, not whether saving part of the solution was worthwhile — only a later task can show that. So put the later task inside the training example: two related tasks in one rollout [SAGE], or one long trajectory [AgeMem].

TASK 1 REWARD $r_1 + 1[\text{both succeed}] \cdot 1[\text{task 2 reused its skill}]$



SAGE · Wang et al., ACL 2026 — AppWorld, Qwen2.5-32B ↗ AgeMem · Yu et al., ACL 2026 ↗

Discussion: what should persist?

exact episode	↔	**general skill**
flexible guidance	↔	**committed execution**
reuse what exists	↔	**explore and replace**
grow the memory	↔	**update, merge, delete**

1. What would you store as an episode, a fact, a lesson, a text skill, or code — and why?
2. When should an agent delete a skill? Revise it?
3. What's the right balance between agent and human effort in creating and using skills?

Ownership of memory decisions

DECISION	HUMAN EFFORT IS VALUABLE WHEN...	AGENT EFFORT IS VALUABLE WHEN...
Author or induce	the skill or policy is already known	it must be discovered through interaction
Verify and admit	correctness is normative or high-stakes	outcomes are executable and cheap to test
Select and use	rare exceptions need contextual judgment	routing repeats and produces feedback
Maintain	accountability requires an owner	drift is visible in outcomes and repairable

Questions?

Next Class: Planning