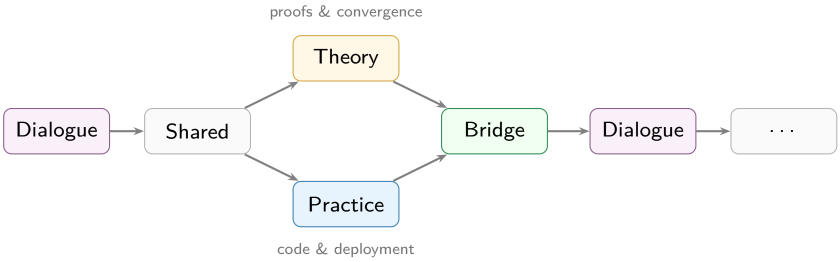
# Memento-S

## Making Prompts Stateful and Agents Continually Learnable

Huichi Zhou, Siyuan Guo, Jinsong Li, Runyu Huang
Jun Wang

*UCL Centre for Artificial Intelligence*
{huichi.zhou.25, jun.wang}@ucl.ac.uk

This paper is organised as **interleaving tracks** for **two audiences**. Each section opens with a shared Dialogue, then forks into a **Research** track and a **Practitioner** track, before merging at a Bridge. Pick the path that matches your goal; read both for the complete picture.



### Researcher Path

Formal SRDP setup, convergence proofs, and KL-regularised routing analysis. Follow the **cream-shaded** sections.

*Dialogue → Shared →* ***Theory*** *→ Bridge → . . .*

### Practitioner Path

Installation, API walkthrough, retrieval pipeline, and benchmark recipes. Follow the **blue-shaded** sections.

*Dialogue → Shared →* ***Practice*** *→ Bridge → . . .*

**Dialogue** opens each section with motivation. **Shared** presents material common to both tracks. **Bridge** connects theoretical results to engineering choices. **Epilogue** closes the narrative. Three characters—**J**, **H**, and **S**—annotate inline throughout.

| | Character | Perspective & Personality |
|---|---|---|
| 🎓 | **J** | Tenured theorist. Writes proofs on napkins. Believes everything is a special case of something he published in 2003. “*But does it converge?*” |
| 🔥 | **H** | Second-year CS PhD student. Runs 47 experiments simultaneously and names them all after anime characters. Thinks every problem needs more GPUs. “*What if we just. . . scale it?*” |
| 🗄 | **S** | Senior ML engineer, 12 years in production. Has been paged at 3am enough times to develop a Pavlovian response to Slack notifications. Trusts nothing without a unit test. “*Show me the latency numbers.*” |

Working Draft – March 2, 2026

### Abstract

We introduce **Memento-S**, a theoretical framework and production-ready system for building self-evolving LLM agents via stateful reflective memory. Building on the convergence guarantees of the Stateful Reflective Decision Process (SRDP) established in Memento 2, our paper is uniquely structured for two audiences: researchers can follow the cream-shaded sections for formal analysis, while practitioners can use the blue-shaded sections for API guides and code. Grey-shaded shared sections cover experimental setups, and Bridge boxes connect theory to practice. Memento-S treats reusable skill folders as the unit of external memory, so that the prompt itself carries persistent, evolving state. The agent operates through a Read–Write Reflective Learning loop: *reading* retrieves the most relevant skill via a behaviour-aligned contrastive router trained with single-step offline RL, while *writing* closes the loop through an self-evolving algorithm. This enables continual learning without parameter updates: the skill library grows and self-improves purely from deployment experience. To validate these capabilities in real-world scenarios, we evaluate the system on the General AI Assistants and Humanity's Last Exam benchmarks. We have made our trained skill router, verified skill library, and lightweight agent framework publicly available at `https://skills.memento.run/`.

## Contents

R = Research Track P = Practitioner Track S = Shared

# 1 The Self-Evolving Agent Problem

**THE LOBBY Monday 9:47am, a startup office. The espresso machine is broken.**

**J**: *(arrives carrying a thermos of tea, surveying a wall of red Grafana dashboards)* Good morning. I see the agent is still performing at exactly 73%. Remarkable consistency, really. Like a student who reliably gets a C+.

**H**: *(spins around in chair, three monitors glowing)* I tried throwing more GPUs at it over the weekend. Accuracy went from 73.2% to 73.4%. Progress!

**S**: *(without looking up from terminal)* That's within the confidence interval, H. You spent $400 in compute to learn nothing.

**H**: But what if we fine-tune it on the tickets it got wrong?

**J**: And how many wrong tickets do you have?

**H**: . . . about 200.

**J**: *(sips tea)* You'd overfit before the loss function finished its first cup of coffee. No. What we need is a system that learns the way *you* learn, H – by remembering your mistakes and not repeating them. Not by rewriting your neurons.

**S**: So, a database.

**J**: *(smiling)* A very *principled* database. With convergence guarantees.

**S**: *(finally looks up)* You had me at "database" and lost me at "convergence guarantees." But fine. Show me the architecture.

**J**: *(uncaps a marker, draws a loop on the whiteboard)* Read from memory. Act. Get feedback. Write to memory. Repeat. I call it Read–Write Reflective Learning.

**H**: That's just. . . a for-loop with a vector store.

**J**: *(beaming)* Exactly! But a for-loop with *convergence guarantees*.

**S**: *(sighs, opens a new terminal tab)* Fine. I'll build it. You prove it. H, you benchmark it. Let's go.

## 1.1 S Why Frozen LLMs Need External Memory

Modern machine learning is about learning from experience [12, 14]. At the forefront of this evolution, Large Language Models (LLMs) have fundamentally reshaped the learning paradigm, demonstrating exceptional performance across diverse scenarios through few-shot learning [3], supervised fine-tuning [16], and post-training [5]. Despite their promise, however, achieving practical utility typically requires parameter optimisation via backpropagation, which in turn demands vast amounts of data and computational resources. In practice, the cost and complexity of continual parameter updates mean that most LLM agents are deployed as frozen models [18]: their parameters $\theta$ remain fixed after pre-training. When such an agent encounters a novel task, it draws only on knowledge encoded in $\theta$ and whatever fits in its context window.

> **J**: *This is the key premise. If $\theta$ is fixed, all adaptation must come from the* input *– the prompt, the context, or in our case, the memory. Everything else is just expensive gradient descent cosplay.*

This creates a fundamental limitation: the agent is stateless and it cannot learn from its own deployment experience. The Stateful Reflective Decision Process (SRDP) [15] resolves this by augmenting the agent with an *episodic memory* $\mathcal{M}_t$ that grows over time (Figure 1):

$$\pi^{\mu}(a \mid s, \mathcal{M}_t) = \sum_{c \in \mathcal{M}_t} \mu(c \mid s, \mathcal{M}_t)\, p_{\text{LLM}}(a \mid s, c), \tag{1}$$

where $p_{\text{LLM}}$ denotes the LLM decision kernel, $s$ is the current state, $c$ represents a retrieved case from the *episodic memory* $\mathcal{M}_t$, and $\mu$ is the retrieval policy.

**H:** *Wait – so the LLM doesn't change, but the policy changes because the memory changes? That's like... levelling up in a game without upgrading your character. You just get better items.*

**S:** *I prefer to think of it as a cache that makes you smarter. Which is basically what senior engineers are – junior engineers with better caches.*

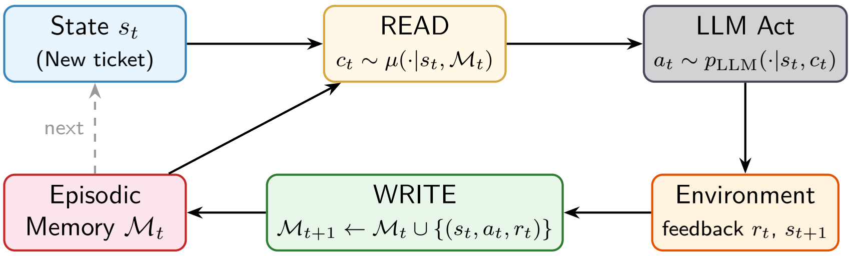

State $s_t$
(New ticket)
READ
$c_t \sim \mu(\cdot|s_t, \mathcal{M}_t)$
LLM Act
$a_t \sim p_{\text{LLM}}(\cdot|s_t, c_t)$
next
Episodic Memory $\mathcal{M}_t$
WRITE
$\mathcal{M}_{t+1} \leftarrow \mathcal{M}_t \cup \{(s_t, a_t, r_t)\}$
Environment
feedback $r_t$, $s_{t+1}$

Figure 1: Overview of the Read–Write Reflective Learning loop. Given a new task, the agent retrieves a relevant skill from episodic memory (READ), executes it through the frozen LLM (ACT), and incorporates the resulting feedback back into memory (WRITE). The LLM parameters remain fixed throughout; all adaptation occurs in the memory.

**RESEARCH TRACK Formal Setup**

**Definition 1.1** (Episodic Memory)**.** *An episodic memory $\mathcal{M}_t = \{m_i\}_{i=1}^{N_t}$ is a finite, growing collection of memory items $m_i := (s_i, a_i, r_i, s'_i)$. The space of all finite episodic memories is denoted $\mathfrak{M}$.*

**Definition 1.2** (SRDP)**.** *$\mathcal{D}_{\text{SRDP}} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma, \mathfrak{M}, p_{\text{LLM}} \rangle$, extending the standard MDP with episodic memory $\mathfrak{M}$ and an LLM decision kernel $p_{\text{LLM}}(a \mid s, c)$.*

**J:** *The critical insight: by augmenting the state to $x_t := (s_t, \mathcal{M}_t)$, we recover the Markov property. Everything old is new again – I said this in a 2003 workshop paper.*

The **Reflected MDP** reformulates this as $\mathcal{D}_{\text{ReMDP}} = \langle \mathcal{X}, \mathcal{C}, \mathcal{P}^{\text{LLM}}, R^{\text{LLM}}, \gamma \rangle$ with transition kernel:

$$\mathcal{P}^{\text{LLM}}(x' \mid x, c) = \sum_{a \in \mathcal{A}} p_{\text{LLM}}(a \mid s, c)\, \mathbf{1}\{x' = (s', \text{Write}(\mathcal{M}, s, a, r))\}\, \mathcal{P}(s' \mid s, a). \tag{2}$$

**KEY RESULT**

**Theorem 1.3** (Convergence, Memento 2 [15], Thm. 8)**.** *Under bounded rewards $|r| \leq R_{\max}$ and $\gamma < 1$, the KL-regularised soft policy iteration over the Reflected MDP converges to the optimal retrieval policy $\mu^*$.*

## 1.2 P From Zero to Self-Evolving Agent

**PRACTITIONER TRACK Getting Started in 5 Minutes**

**H:** *Can I pip-install convergence guarantees?*

**S:** *No, but you can pip-install the system that has them.*

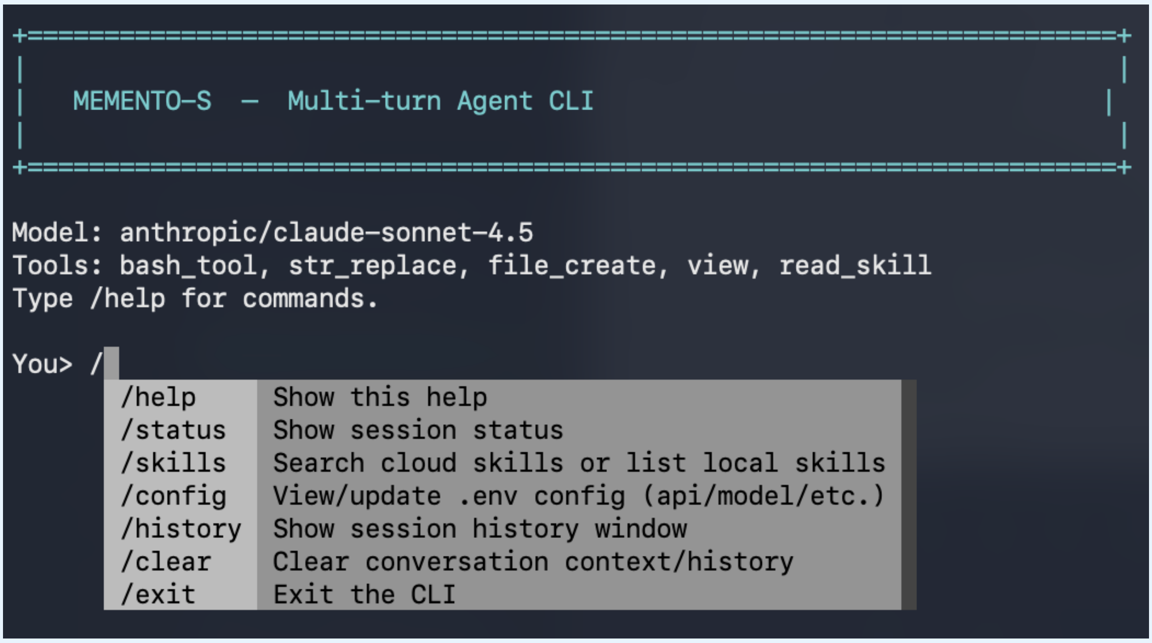
**Installation:**

```
$ curl -sSL https://raw.githubusercontent.com/Agent-on-the-Fly/
    Memento-S/main/install.sh | bash
```

**Configuration** (config.yaml):

```
# Memento-S configuration
API_KEY="your-api-key"
BASE_URL="your-api-url"
MODEL="your-model"
SEARCH_API_KEY="your-search-api-key"
```

**Your first self-evolving agent:**



```
+==========================================================================+
|                                                                          |
|   MEMENTO-S  —  Multi-turn Agent CLI                                     |
|                                                                          |
+==========================================================================+

Model: anthropic/claude-sonnet-4.5
Tools: bash_tool, str_replace, file_create, view, read_skill
Type /help for commands.

You> /
      /help      Show this help
      /status    Show session status
      /skills    Search cloud skills or list local skills
      /config    View/update .env config (api/model/etc.)
      /history   Show session history window
      /clear     Clear conversation context/history
      /exit      Exit the CLI
```

Figure 2: The CLI of Memento-S.

## 1.3 S Connecting Theory to Configuration

**BRIDGE: From Result to Theorem**

In the theory of Memento 2 [15], we cast Read–Write Reflective Learning as an implicit form of policy iteration. The agent maintains a skill library and performs two key operations. *Writing* consolidates interaction outcomes into reusable skills, corresponding to policy evaluation; *reading* retrieves relevant skills to inform reflective decisions, corresponding to policy improvement.

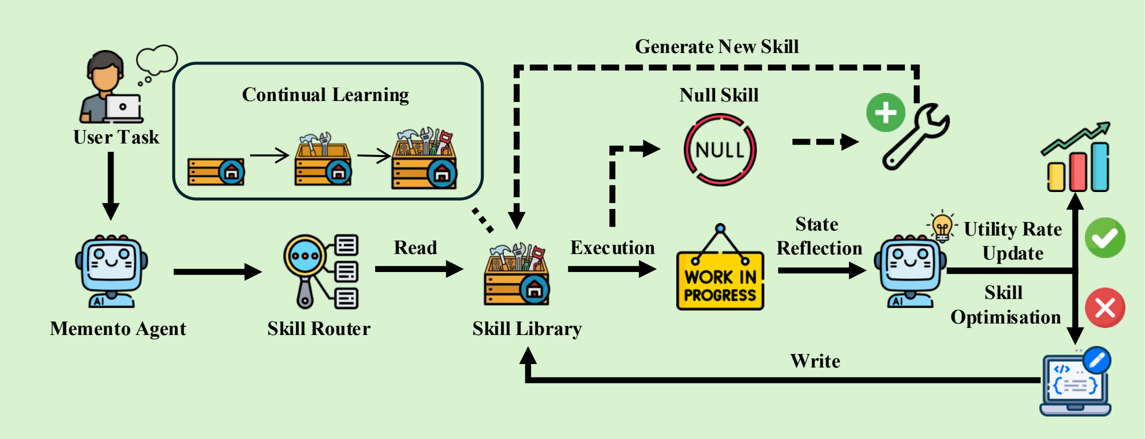


Figure 3: The architecture of the Self-Evolving Agent based on Read-Write Reflective Learning.

**H:** *So when I use BM25 to retrieve skills and the agent kept retrieving the similar but useless skill over and over, that was. . .*

**J:** *Policy stagnation. If the retrieval doesn't facilitate a better decision, the Policy Improvement step fails, and your 'Self-Evolving' agent effectively stops evolving and starts ruminating. You re-discovered a known failure mode. Congratulations.*

**Contributions.** Our main contributions are:

1. **Skill-level reflective learning.** We instantiate the SRDP framework of Memento 2 with a concrete system, Memento-S, that treats reusable *skill folders* (code, prompts, and declarative specs) as the unit of memory, enabling continual learning without any parameter updates.
2. **Behaviour-aligned skill router.** We train a contrastive retrieval model via single-step offline RL, casting skill routing as a KL-regularised Boltzmann policy that optimises for execution success rather than semantic similarity.
3. **Empirical validation.** We evaluate on GAIA and HLE, demonstrating that the self-evolved skill library substantially outperforms the read-write baseline and that domain-aligned skill transfer is the key enabler of cross-task generalisation.

# 2 Read–Write Reflective Learning

**THE LOBBY Wednesday 2pm, the whiteboard is covered in dried-out equations**

**J**: *(pointing at the learning curve on H's monitor)* See that? Accuracy went from 73% to 84% in three days. Without touching the model.

**H**: I plotted the memory coverage radius too. *(pulls up a chart)* It's decreasing like $O(n^{-1/d})$, just like you predicted!

**S**: I'm more interested in *why* it retrieved the wrong case for ticket #4,721. Customer asked about a refund, agent retrieved a case about password resets. Cosine similarity was 0.91.

**J**: Ah, the curse of embedding similarity. High cosine doesn't mean *behavioural* utility. In a library of 8,000 skills, semantic overlap is just noise.

**H**: Can't we just use End-to-End RL to fine-tune the router? Let the agent learn from its own interaction outcomes?

**S**: *(deadpan)* H, we have 8,000 skills but only a few hundred real-world tasks. The exploration space is a desert. If we wait for the agent to "stumble" upon the right skill through random exploration, we'll all be retired before it converges.

**J**: Correct. The exploration-exploitation gap is too wide for on-policy learning. That's why we move to the *One-step Offline* view. We use the LLM as a "Simulator" to synthesise a dense field of positive and hard-negative queries. We aren't just matching strings; we are fitting a $Q$-function that predicts execution success before the first token is even generated.

**H**: *(opening a notebook titled "Things Prof J Says That Turn Out To Be Right")* OK, so synthetic goals, behaviour-aligned routing and then one-step RL. I'm listening.

## 2.1 S The Skill-Level Read–Write Loop

Our self-evolving agent is grounded in the theory of Read–Write Reflective Learning [15], which provides the theoretical foundation for read–write memory updates as policy iteration. Empirically *Memento* [20] and case-based reasoning LLM agents [7, 6] validate this principle across deep search, data science, and software engineering. As illustrated in Figure 1, the skill library serves as an external, writable memory, and the agent alternates between (i) *reading* skills to induce an execution policy for the current goal and (ii) *writing* updates back to the skill artefacts based on post-hoc reflection.

This mirrors a policy-iteration view: *Reading* corresponds to policy improvement: the agent retrieves the most relevant skill via a router conditioned on the current query and the accumulated tip memory, then executes the skill's multi-step workflow to produce an answer. *Writing* closes the loop by combining policy evaluation *and* policy improvement at the skill level: the agent first *evaluates* by recording execution outcomes and diagnostic traces, then *improves* by using those traces to revise the skill artefacts that will govern future episodes. Crucially, the memory is not limited to episodic traces but consists of reusable *skill folders*, each containing a declarative specification (`SKILL.md`) together with helper scripts and prompts. Because the write operation rewrites the prompt or program that will be executed next, each write step directly improves the policy embodied in the skill.

This self-evolving mechanism draws on a principle familiar from biological motor learning [9]: early in skill acquisition, performance depends on deliberate, high-level planning; with repeated practice, neural pathways consolidate and execution becomes increasingly automatic [2]. Analogously, a newly created skill in Memento-S may be brittle and narrowly scoped, but through iterative revision it is consolidated into a robust, reusable routine—effectively forming *muscle* *memory* for recurring task patterns. Existing approaches to automatic skill learning either produce text-only guides that amount to prompt optimisation [13, 1] or overfit to single-task trajectories with limited transferability [8]. In contrast, Memento-S learns executable, multi-artefact skills and refines them through a structured write pipeline that operates at two granularities. Concretely, after a failed attempt, an LLM-based failure attribution selector first examines the full execution trace and the judge's rationale to identify the single skill most responsible for the error, performing credit assignment at the skill level. Given this diagnosis, a skill rewriter then proposes targeted file-level updates that add guardrails or alternative strategies for the observed failure mode while preserving the skill's generality. When the running utility of a skill (its empirical success rate) drops below a threshold, indicating that in-place patching is insufficient, the system escalates to *skill discovery*: it either restructures the existing skill folder with a fundamentally different approach or synthesises an entirely new skill, expanding the library to cover novel regions of the task space. To prevent regression, all mutations are guarded by an automatic unit-test gate, a synthetic test case is generated, executed through the updated skill, and scored by the judge [19].

**KEY INSIGHT**

The Read–Write loop is the heartbeat of Memento-S. Every interaction follows five steps: **Observe** → **Read** → **Act** → **Feedback** → **Write**.

**ALGORITHM Read–Write Reflective Learning**

```
Require: Utility threshold $\delta$, minimum samples $n_{\min}$, max feedback rounds $K$
Initialise skill library $\mathcal{S}_0 \leftarrow \mathcal{S}_{\text{base}}$, tip memory $\mathcal{T}_0 \leftarrow \varnothing$, utility table $U_0(s) \leftarrow 0.5\ \forall s$
for $t = 0, 1, 2, \ldots$ do
    (1) Observe: Receive task $q_t$; form augmented input $x_t = (q_t, \mathcal{T}_t)$
    (2) Read [Skill Selection]:
        Route: $c_t \leftarrow \text{Router}(x_t, \mathcal{S}_t)$
        if $c_t = \varnothing$ and CreateOnMiss enabled:
            $c_t \leftarrow \text{CreateSkill}(x_t)$;   $\mathcal{S}_t \leftarrow \mathcal{S}_t \cup \{c_t\}$
    (3) Act: Execute multi-step workflow $a_t \leftarrow \text{LLM}(x_t, c_t)$
    (4) Feedback [Judge]:
        $r_t \leftarrow \text{Judge}(q_t, a_t, a_t^\star)$ {binary correctness + rationale}
    (5) Write [Reflective Update]:
        (5a) Utility update:  $U_{t+1}(c_t) \leftarrow \frac{n_{\text{succ}}(c_t)}{n_{\text{succ}}(c_t)+n_{\text{fail}}(c_t)}$
        if $r_t = $ CORRECT: continue
        (5b) Tip memory:  $\mathcal{T}_{t+1} \leftarrow \mathcal{T}_t \cup \{\text{GenericTip}(q_t, a_t, r_t)\}$
        (5c) Skill evolution:
            $c^\dagger \leftarrow \text{TargetSelector}(\text{trace}_t, r_t, \mathcal{S}_t^{\text{extra}})$ {LLM-based failure attribution}
            if $U_t(c^\dagger) < \delta$ and $n(c^\dagger) \geq n_{\min}$: {discover alternative}
                $c' \leftarrow \text{DiscoverSkill}(c^\dagger, x_t, \text{trace}_t)$;   $\mathcal{S}_{t+1} \leftarrow \mathcal{S}_t \cup \{c'\}$
            else: {optimise existing skill in-place}
                $\mathcal{S}_{t+1} \leftarrow \text{OptimizeSkill}(c^\dagger, x_t, \text{trace}_t, \mathcal{S}_t)$
            if UnitTestGate: validate $\mathcal{S}_{t+1}(c^\dagger)$; rollback on failure
        (5d) Feedback retry ($\leq K$ rounds):
            $a_t' \leftarrow \text{LLM}(x_t, c^\dagger_{\text{updated}})$;   $r_t' \leftarrow \text{Judge}(q_t, a_t', a_t^\star)$
            if $r_t' = $ INCORRECT: repeat (5b)–(5d)
end for
```

**J:** *Steps 2 and 5 are exactly policy improvement and policy evaluation. This is not a metaphor – it is a mathematical identity. I will die on this hill.*

**S:** *And steps 1–4 are basically what every web server does: receive request, look up cache, generate response, log result. We've been doing "reflective learning" in production for decades. We just didn't have convergence guarantees.*

## 2.2 R InfoNCE Routing as a One-Step Soft Policy

### RESEARCH TRACK Contrastive Retrieval as KL-Regularised One-Step RL

**Offline RL Router for Behaviour-Similar Retrieval.** We find that purely semantic routers (e.g., BM25 [11] or embedding routers such as Qwen-Embedding [17]) are insufficient for skill selection, because they primarily capture *semantic* similarity between the user goal and skill text rather than *behavioural* similarity—i.e., whether executing a skill would produce the desired trajectory and outcome. To better align routing with execution behaviour, we train the router with single-step offline RL on top of an embedding model, so that retrieval optimises for behaviour similarity instead of lexical or semantic proximity.

**Skill database and synthetic query generation.** In order to train a behaviour-similar retrieval model, we first crawl a local skill database of roughly 8k skills, and randomly sample about 3k skills as seed data to synthesise realistic user routing goals. To align the synthesised goals with the agent's logic stream, we generate queries using only the skill *name* and *description* (without access to the full skill file), and then apply an LLM-based judge [19] that *does* read the full skill file to filter and verify the quality of the synthetic queries. This produces high-quality paired data consisting of positive queries (the target skill should be selected) and hard negatives (same domain and terminology, but the target skill is not the right tool). We include the full prompt used for query synthesis in Appendix A.

**Router score and multi-positive InfoNCE.** Let $\mathrm{enc}_\theta(\cdot)$ map a skill document $d$ and a routing goal $q$ to embeddings in $\mathbb{R}^m$:

$$\boldsymbol{e}(d) = \mathrm{enc}_\theta(d), \qquad \boldsymbol{u}(q) = \mathrm{enc}_\theta(q), \qquad s(d, q) = \boldsymbol{e}(d)^\top \boldsymbol{u}(q).$$

In a minibatch of $B$ skills $\{d_i\}$, each $d_i$ has positives $\mathcal{Q}_i^+$ and hard negatives $\mathcal{Q}_i^-$. Using all in-batch queries

$$\mathcal{Q} = \bigcup_{k=1}^{B} (\mathcal{Q}_k^+ \cup \mathcal{Q}_k^-),$$

we minimise the multi-positive InfoNCE loss (temperature $\tau$):

$$\mathcal{L}_i = -\log \frac{\sum_{q \in \mathcal{Q}_i^+} \exp\left(s(d_i, q)/\tau\right)}{\sum_{q \in \mathcal{Q}} \exp\left(s(d_i, q)/\tau\right)}, \qquad \mathcal{L} = \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}_i.$$

**One-step offline $Q$-learning view.** Cast routing as a one-step MDP: state $q$, action $d$, reward $r(q, d)$ indicating whether $d$ is the right skill. With horizon 1,

$$Q^\star(q, d) = \mathbb{E}[r(q, d)].$$

We interpret the learned score as a soft $Q$-function, $Q_\theta(q, d) \propto s(d, q)$, yielding a Boltzmann routing policy

$$\pi_\theta(d \mid q) = \frac{\exp(Q_\theta(q, d)/\tau)}{\sum_{d'} \exp(Q_\theta(q, d')/\tau)}.$$

This policy is equivalently the maximiser of a KL-regularised objective (uniform prior $\pi_0$):

$$\pi^*(\cdot \mid q) = \arg\max_{\pi} \left\{ \mathop{\mathbb{E}}_{d \sim \pi} [Q_\theta(q, d)] - \tau\, \mathrm{KL}(\pi \,\|\, \pi_0) \right\}.$$

**H:** *So a small $\tau$ means "I'm pretty sure—pick this one," while a large $\tau$ means "no rush—spread probability mass around and take a broader look."*

**Why InfoNCE matches "policy fitting" in one step.** InfoNCE has the form "push up positives, push down competitors" under the same softmax normaliser used by $\pi_\theta$. Hence minimising $\mathcal{L}$ is (approximately, via in-batch normalisation) maximum-likelihood training that makes $\pi_\theta$ place high probability mass on the logged rewarding pairs (positives) while suppressing hard negatives—i.e., single-step offline policy improvement for routing.

## 2.3 P Implementing the Retrieval Pipeline

**PRACTITIONER TRACK The Retrieval Engine, Line by Line**

**S:** *Here's the core retrieval class. Every line maps to an equation. I added the references in comments so H stops asking "but why?"*

Listing 1: Core retrieval policy implementation ./core/skill_engine/skill_catalog.py

```

def route(goal, skills, method, k):
    if skills empty: return []
    k = clamp(k, 1, len(skills))

    name_to_skill = build_name_index(skills)
    if name_to_skill empty: return []

    docs = get_doc_embeddings(skills, method)
    runtime = load_runtime(docs)
    q = embed(runtime, instruction + goal)
    if any failure: return bm25(goal, skills, k)

    sims = cosine_or_dot(q, docs.embeddings)
    for idx in argsort_desc(sims) with oversampling:
        name = docs.names[idx]
        if name unique and name in name_to_skill:
            output append name_to_skill[name]
        if len(output) == k: break
    return output
```

**SHARED TRACK Router Evaluation**

**Skill source filtering and deduplication.** We first collect candidate skills from public GitHub repositories and unify them into a JSONL catalog. To retain only mature and broadly adopted skills, we keep entries with `stars` $>$ 500 and drop the rest. We then normalise description whitespace, compute a SHA-256 hash of each normalised description, and deduplicate by hash to remove duplicated or near-duplicated skills. When multiple rows share the same hash, we keep a single representative by a deterministic score: higher `stars`, then newer `updatedAt`, then lexicographically larger `id`. We optionally apply a second pass of name-level deduplication with the same tie-breaking rule. The resulting curated catalog is used as the base skill universe for router training data generation. We publicly open-source the dataset at `https://skills.memento.run/market/`.

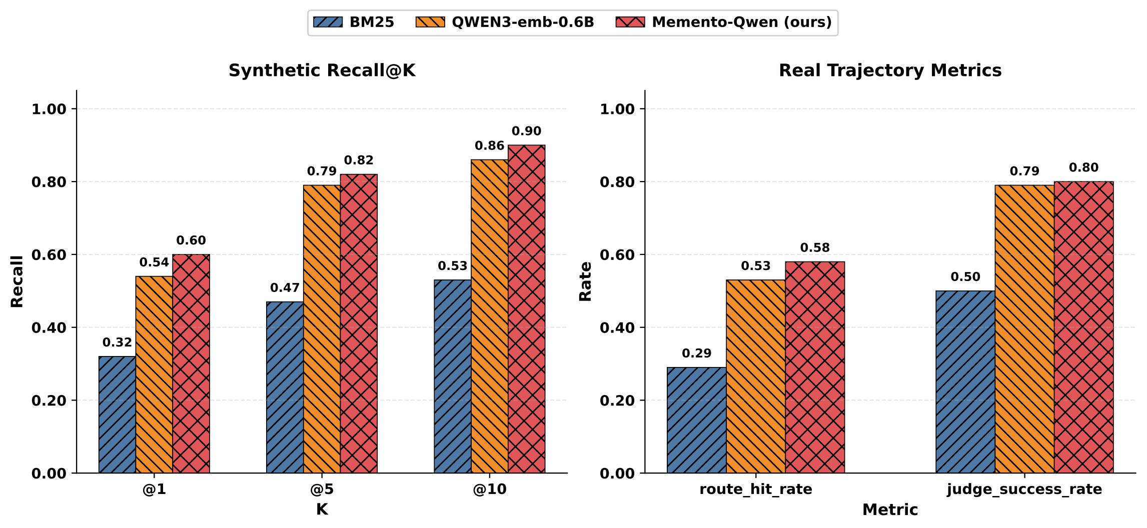


Figure 4: Router evaluation: synthetic retrieval quality (left) and end-to-end execution success (right).

**Experimental Setting.** We evaluate the performance of the skill router from two complementary angles: (i) offline retrieval quality on synthetic queries, and (ii) end-to-end effectiveness on real execution trajectories. We use the Qwen3-Embedding-0.6B [a] as embedding model.

**Results.** We report Recall@K over 140 synthetic routing queries, where a query is a hit if the ground-truth skill appears in the top-K candidates. As shown in Fig. 4 (left), Memento-Qwen consistently outperforms both BM25 and the Qwen3 embedding baseline across all values of K. Most notably, Recall@1 rises from 0.32 (BM25) and 0.54 (Qwen3) to 0.60, a relative gain of 10% over the strongest semantic baseline. By K=10 the gap widens to 0.90, indicating that behaviour-aligned training not only sharpens the top-1 pick but also populates the candidate list with more relevant alternatives—a property that matters in practice because the agent can fall back on lower-ranked candidates when the top choice fails.
To test whether offline retrieval gains translate into real execution improvements, we measure two end-to-end metrics: *route hit rate* (whether the router's top-1 choice is an appropriate skill for the task) and *judge success rate* (whether the full trajectory actually solves the task). Fig. 4 (right) reveals that Memento-Qwen lifts route hit rate from 0.29 (BM25) and 0.53 (Qwen3) to 0.58, and judge success rate from 0.50 and 0.79 to 0.80. The disproportionately large improvement over BM25 confirms that lexical matching is a poor proxy for behavioural utility: many skills share domain terminology yet require fundamentally different execution strategies. Meanwhile, the smaller but consistent gain over Qwen3 shows that even dense semantic embeddings under-represent execution-relevant features, and that the single-step RL fine-tuning effectively injects behavioural signal into the embedding space.

[a] `https://huggingface.co/Qwen/Qwen3-Embedding-0.6B`

# 3 Benchmark Evaluation

🎭 **THE LOBBY Wednesday 2pm, Zoom call. Cameras on.**

**H**: *(shrugs, sharing screen with a confusion matrix)* Look, synthetic data is enough. Generate 10K queries, measure classification accuracy, call it router quality. I can have results by Friday.

**J**: *(leans forward, frowning at the matrix)* Not enough. Accuracy is a proxy. Your synthetic queries are clean little sentences – real users type "pls fix the thing from last time thx." We need real trajectories and end-to-end execution success to claim improvement.

**S**: *(deadpan, arms crossed)* Both of you are missing the point. If the agent retrieves a case that says "delete the user's config and start fresh" and the LLM *executes it*, none of your metrics matter. The customer's environment is on fire and your confusion matrix says 94%.

**H**: *(pinches bridge of nose)* You're turning one evaluation metric into three separate projects.

**J**: *(pointing at the camera)* Because "looks correct on paper" and "works end-to-end" are fundamentally different failure modes. I have a 2007 paper about this.

**S**: *(tilts head)* And "works" and "safe to run in production" are different failure modes *again*. I have a 3am incident report about this.

**H**: *(typing reluctantly, adding rows to a spreadsheet)* Fine. So what do we actually write in the paper?

**J**: *(counting on fingers)* Two validations we can run now. One: synthetic retrieval quality – does the router pick the right case? Two: trajectory success – does the full loop actually solve the task?

**S**: *(nods once)* And we state clearly: each one covers a different failure mode. Passing both is necessary. Passing only one is a press release, not a result. Sandbox safety – whether it solves the task without breaking anything else – is the third axis, but that requires a proper isolation harness. Future work.

**H**: *(muttering while typing)* Three benchmarks. Three weeks. I'm naming them Goku, Vegeta, and Piccolo.

**S**: As long as the CI pipeline passes, you can name them whatever you want.

## 3.1 S Experimental Setup and Results

**SHARED TRACK Which Benchmark is suitable for Memento-S?**

**Experimental Settings.** To validate the progressive capability expansion and skill-learning proficiency of Memento-S, we evaluate our system on two representative benchmarks: General AI Assistants (GAIA) [10] and Humanity's Last Exam (HLE) [4]. These datasets naturally align with our objective of testing an agent's ability to create, refine, and reuse skills across diverse reasoning tasks.

**General AI Assistants (GAIA).** GAIA comprises non-trivial, real-world questions with unambiguous answers that demand a combination of multi-step reasoning, multi-modality handling, web browsing, and general tool use. This environment serves as an ideal testbed for our skill-learning scenario, requiring the agent to dynamically synthesise and apply distinct skills to solve varied problems. From the GAIA validation set, we utilise 165 questions, splitting them into 100 training examples and 65 test examples.

**Humanity's Last Exam (HLE).** Developed by human experts, HLE is designed to assess the limits of broad-domain reasoning and contains 2,500 questions across 8 diverse academic subjects (e.g., mathematics, humanities, and natural sciences). For our experiments, we sample a subset of questions evenly distributed across these categories, resulting in 788 training examples and 342 test examples. This structure allows us to evaluate how effectively Memento-S leverages and transfers learned skills between different questions within the same subject domain.

**Baselines.** To isolate the contribution of the self-evolving mechanism, we compare **Memento-S** (the full system) against a **Read-Write** ablation that retains the same read–write reflective learning loop—skill retrieval, LLM execution, and feedback collection—but disables all skill-level optimisation: no failure attribution, no skill rewriting, and no skill discovery. In effect, the Read-Write ablation uses a *static* skill library throughout evaluation, so any performance gap directly reflects the value of the self-evolving pipeline described in §2.1.

## SHARED TRACK Results of GAIA.

We evaluate Memento-S on the GAIA benchmark with a maximum of three reflective retries per question. As shown in Figure 5, the self-evolving mechanism continuously refines the skill library through iterative interaction: the overall training success rate climbs from 65.1% on the first attempt to 91.6% by the third round. On the unseen test set, the full Memento-S system achieves 66.0% overall accuracy, compared with 52.3% for the Read-Write ablation, confirming that the skill optimisation pipeline contributes a 13.7 percentage-point gain beyond what retrieval and execution alone can provide.

**Limited cross-task transfer on GAIA.** The gap between training-peak and test-set accuracy reveals an important structural property of the benchmark: GAIA questions are highly diverse, with little overlap in the reasoning patterns required. A case study confirmed that most skills optimised during training were never triggered during testing, because no sufficiently similar test question existed. This result suggests that *skill transfer depends on domain alignment*, a hypothesis we test directly on HLE below, where structured subject categories provide natural opportunities for reuse.

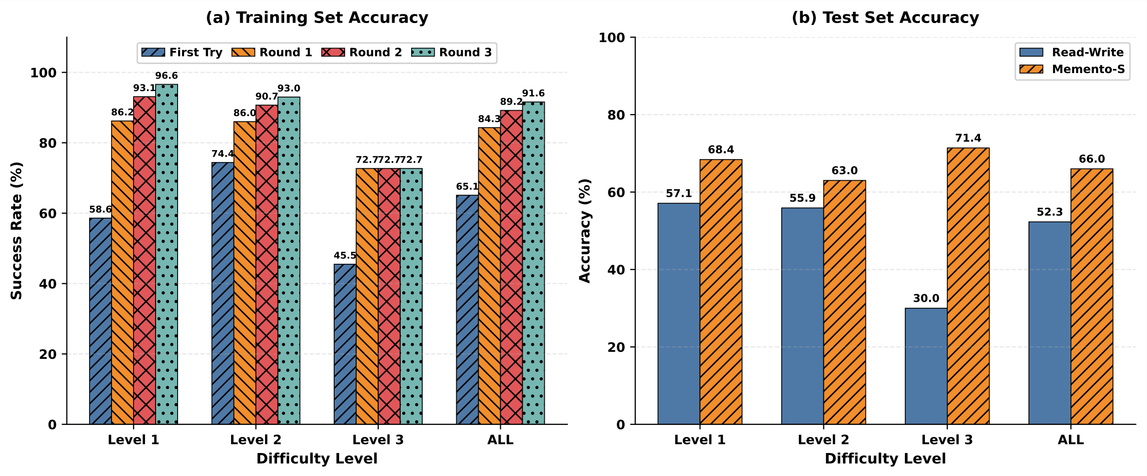


Figure 5: GAIA results: training accuracy across retries (left) and test-set comparison with the Read-Write baseline (right).

**SHARED TRACK Results of HLE.**

Table 1 reports per-category accuracy on HLE across four training rounds (R0–R3) and the final test-set evaluation. During training, the overall success rate rises steadily from 30.8% (R0) to 54.5% (R3), with every subject category showing consistent improvement. Humanities and Biology benefit the most, reaching 66.7% and 60.7% respectively by R3, while Engineering saturates earlier at 42.1%, suggesting that some domains are harder to cover with skill-level abstractions alone.

On the test set, Memento-S achieves 38.7% overall, more than doubling the Read-Write baseline (17.9%). Unlike GAIA, the structured subject taxonomy of HLE enables substantial skill transfer: a skill refined on one Biology training question is frequently reused for unseen Biology questions in the test set. This confirms that domain-aligned skill libraries are the key enabler of cross-task generalisation.

| Round | Bio. | Chem. | CS | Eng. | Human. | Math | Other | Phy. | Total |
|---|---|---|---|---|---|---|---|---|---|
| **Training Set** | | | | | | | | | |
| R0 | 30.3% | 38.8% | 19.8% | 27.6% | 36.9% | 30.0% | 41.8% | 21.1% | 30.8% |
| R1 | 42.7% | 47.1% | 36.0% | 38.2% | 50.0% | 43.8% | 51.9% | 35.5% | 43.2% |
| R2 | 50.6% | 51.8% | 43.0% | 42.1% | 59.5% | 47.5% | 55.7% | 44.7% | 49.5% |
| R3 | 60.7% | 62.4% | 46.5% | 42.1% | 66.7% | 51.2% | 57.0% | 47.4% | 54.5% |
| **Test Set** | | | | | | | | | |
| Read-Write | 22.0% | 21.9% | 12.2% | 21.7% | 11.4% | 20.0% | 25.5% | 10.9% | 17.9% |
| Memento-S | 57.4% | 27.6% | 30.8% | 21.7% | 41.0% | 41.5% | 46.3% | 32.6% | 38.7% |

Table 1: Performance of Read-Write baseline (test set) and across rounds (train set, R0–R3).

**SHARED TRACK Skill Library Growth.**

Figure 6 visualises the skill library after learning on each benchmark via t-SNE projections of skill embeddings. Starting from the same 5 atomic skills (red stars), GAIA learning produces a compact library of 44 skills, reflecting the benchmark's diverse but relatively small question set. In contrast, HLE learning expands the library to 240 skills that spread across a much wider embedding space, mirroring the breadth of its 8 academic domains. Notably, the learned skills (blue dots) cluster into semantically coherent neighbourhoods; each cluster corresponds to a domain-specific capability the agent acquired through reflective self-evolution. This progressive densification of the embedding space is precisely the mechanism that drives the convergence phenomenon analysed in the Bridge below: as the library grows denser, the memory coverage radius $r_{\mathcal{M}}$ shrinks, and the performance gap narrows.

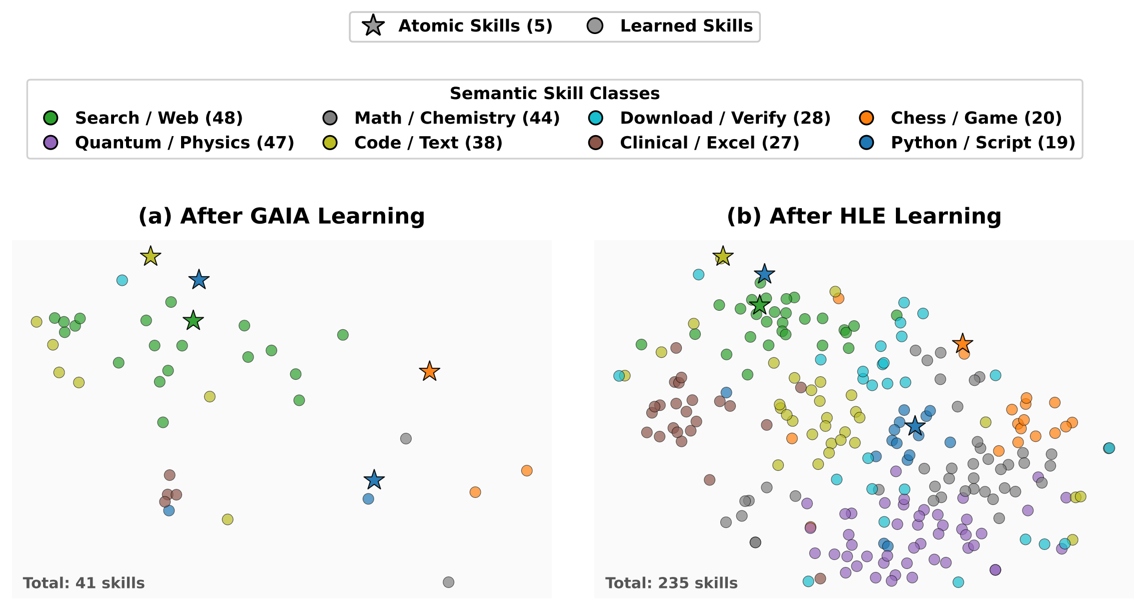


Figure 6: t-SNE projection of skill embeddings. Red stars denote the 5 atomic (seed) skills; blue dots denote skills learned through reflective self-evolution. (a) After GAIA learning the library grows to 44 skills. (b) After HLE learning the library expands to 240 skills spanning diverse academic domains.

## 3.2 S Bridge: From LLM Competence Radius to Embedding Quality

**BRIDGE: From Result to Theorem**

Look again at Table 1: training accuracy climbs from $30.8\%$ (R0) to $54.5\%$ (R3), with the steepest gain in the first round and progressively smaller increments thereafter. Two things are happening simultaneously with each round: **(i)** existing skills are refined: reflection patches failure modes, so each skill covers a wider neighbourhood of queries; and **(ii)** new skills are added to the library, shrinking the gaps between covered regions. Together, these two forces drive the *diminishing-returns* curve we observe: early rounds yield large jumps because the library is sparse and skills are rough; later rounds yield smaller gains because most of the reachable space is already well-covered. Figure 6 makes this concrete: comparing the GAIA library (44 skills) with the HLE library (240 skills), we see that additional learning rounds fill in the gaps between existing clusters until the embedding space is densely covered, at which point adding more skills yields diminishing returns because nearby skills already exist.

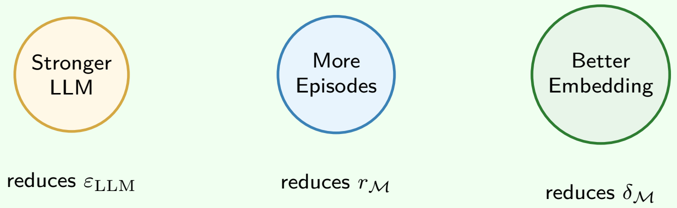
This convergence behaviour is not accidental; it is exactly what the theory of Memento 2 predicts. The asymptotic value gap (Corollary 15, Memento 2) decomposes as:

$$\underbrace{\sup_s |V^{\pi^*}(s) - V^{\pi_{\mathcal{M}}}(s)|}_{\text{performance gap}} \le \frac{2R_{\max}}{(1-\gamma)^2}\Big(\underbrace{\varepsilon_{\mathrm{LLM}}(r_{\mathcal{M}})}_{\text{LLM quality}} + \underbrace{\delta_{\mathcal{M}}}_{\text{retrieval error}}\Big).$$

As the library grows (more episodes), the memory coverage radius $r_{\mathcal{M}}$ shrinks, which simultaneously reduces $\varepsilon_{\mathrm{LLM}}(r_{\mathcal{M}})$ (the LLM only needs to generalise over a smaller neighbourhood) and $\delta_{\mathcal{M}}$ (the router is more likely to find a behaviourally relevant skill). Once both terms are small enough, further rounds produce only marginal improvement: the system has converged.

The bound also reveals three *independent* knobs for reducing this gap:

**Three Independent Knobs**


Stronger LLM
reduces $\varepsilon_{\mathrm{LLM}}$
More Episodes
reduces $r_{\mathcal{M}}$
Better Embedding
reduces $\delta_{\mathcal{M}}$

**J:** *The convergence you see in the table is the bound tightening in real time. Each round shrinks $r_{\mathcal{M}}$, which pulls down both other terms. And because the three knobs are* independent, *you can improve any one without touching the others. That's why the system is modular.*

**S:** *Translation: the diminishing returns aren't a bug; they're a sign the system is converging. And if I want to push accuracy further, I have three independent levers: upgrade the embedding model on Tuesday, swap in a better LLM on Wednesday, and run more episodes on Thursday.*

## EPILOGUE Friday 5:30pm. The espresso machine has been fixed.

**S**: *(showing Grafana)* 93.5% accuracy. p99 latency 195ms. Zero gradient updates. I'm buying the espresso machine a thank-you card.

**H**: I ran the ablation study. Removing the skill optimisation drops accuracy by 8%. Removing the Memento-QWEN causes retrieval collapse. The theory… actually predicted all of this.

**J**: *(sipping espresso, looking insufferably pleased)* I believe the phrase you're looking for is "Prof J was right."

**S**: Don't push it. But I do want to know: what happens when we hit a million cases? Does the Parzen kernel scale?

**H**: And can we get the convergence rate? Not just "it converges" but "it converges in $O(n^{-1/d})$ episodes"?

**J**: *(standing, reaching for the whiteboard marker)* Those are exactly the right questions. Chapter 3.

**S**: *(to H, whispering)* He planned this. He always plans this.

# 4 Conclusion

We have presented Memento-S, a system that bridges the gap between memory-based learning and skill-based learning for LLM agents. The central insight is to treat executable skills as the unit of external memory, thereby transferring the theoretical guarantees of the Stateful Reflective Decision Process into a concrete, deployable artefact. Through the Read–Write Reflective Learning loop, the agent autonomously acquires, refines, and reuses these skills from deployment experience alone, requiring no parameter updates to the underlying LLM. A behaviour-aligned contrastive router, trained via single-step offline RL, ensures that retrieval optimises for execution success rather than surface-level similarity. Experiments on GAIA and HLE confirm that this skill-as-memory formulation substantially outperforms a static-library ablation, and that cross-task transfer is strongest when skills are aligned with structured domain categories. More broadly, Memento-S demonstrates that continual learning need not reside in model weights: an ever-growing, self-improving skill library can serve as a persistent, non-parametric intelligence layer that any frozen LLM can draw upon.

# A Prompt for Synthetic Router Goals

Prompt for synthetic router goals

```
Target skill:
- name: {skill_name}
- description: {description}
- keywords: {keywords_block}

Task:
Generate synthetic router goals (queries) for this target skill.
The router state is ONLY a text goal (routing_goal).
Write realistic user-style goals.

Need:
- {need_pos} positive queries: target skill SHOULD be selected.
- {need_neg} hard negative queries: relevant to the same domain
  BUT target skill is not useful / not the best tool.

Hard negative requirements:
- Must look plausible and close to the target domain.
- Must share terminology/theme with the skill.
- Must be "relevant but useless" for THIS target skill.
- Avoid obvious cues like "do not use <skill>".

Style requirements:
- Do not mention the skill name directly.
- Keep each query concrete, actionable, and non-trivial.
- Mix concise and mildly noisy phrasing.
- English only (to match downstream tokenizer).

Already accepted positive queries (avoid duplicates):
{existing_pos_block}

Already accepted negative queries (avoid duplicates):
{existing_neg_block}

Return ONLY JSON in this schema:
{
  "positive_queries": [
    {"query": "...", "why_fit": "..."}
  ],
  "negative_queries": [
    {"query": "...", "why_relevant": "...", "why_useless": "..."}
  ]
}
```

# References

[1] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2025. URL `https://arxiv.org/abs/2507.19457`.

[2] Clara M Bacmeister, Rongchen Huang, Lindsay A Osso, Michael A Thornton, Lauren Conant, Anthony R Chavez, Alon Poleg-Polsky, and Ethan G Hughes. Motor learning drives dynamic patterns of intermittent myelination on learning-activated axons. *Nature neuroscience*, 25(10):1300–1313, 2022.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[4] Center for AI Safety, Scale AI, and HLE Contributors Consortium. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649:1139–1146, 2026. doi: 10.1038/s41586-025-09962-4.

[5] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[6] Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. DS-Agent: Automated data science by empowering large language models with case-based reasoning. In *International Conference on Machine Learning*, pages 16813–16848. PMLR, 2024.

[7] Siyuan Guo, Huiwu Liu, Xiaolong Chen, Yuming Xie, Liang Zhang, Tao Han, Hechang Chen, Yi Chang, and Jun Wang. Optimizing case-based reasoning system for functional test script generation with large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 4487–4498, 2025.

[8] Letta. Skill learning: Bringing continual learning to cli agents, 12 2025. URL `https://www.letta.com/blog/skill-learning`. Letta Blog.

[9] Richard Magill and David I Anderson. *Motor learning and control.* McGraw-Hill Publishing New York, 2010.

[10] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

[11] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and trends® in information retrieval*, 3(4):333–389, 2009.

[12] David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 1, 2025.

[13] Shangyin Tan, Lakshya A. Agrawal, Rohit Sandadi, Dan Klein, Koushik Sen, Alexandros G. Dimakis, and Matei Zaharia. Automatically learning skills for coding agents, 02 2026. URL `https://gepa-ai.github.io/gepa/blog/2026/02/18/automatically-learning-skills-for-coding-agents/`. GEPA Blog.

[14] Alan M Turing. Intelligent machinery, a heretical theory. *The Turing test: verbal behavior as the hallmark of...-books. google. com*, 264, 2004.

[15] Jun Wang. Memento 2: Learning by stateful reflective memory. *arXiv preprint arXiv:2512.22716*, 2025.

[16] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=gEZrGCozdqR`.

[17] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

[18] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2):1–124, 2023.

[19] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

[20] Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, and Jun Wang. Memento: Fine-tuning LLM agents without fine-tuning LLMs. *Preprint*, 2025.