AGENTSPEC: EVOLVING EMBODIED-AGENT HARNESSES THROUGH TYPED COMPOSITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Embodied LLM agents run inside harnesses of perception, memory, reasoning, reflection, and action, and coding agents can now write and revise these harnesses. We present AGENTSPEC, a typed specification and runtime in which each component has a declared contract and a complete harness is an executable, diffable program that a coding agent can modify, compose, run, and trace. With it we compare how one coding agent constructs harnesses for one fixed executor: evolving reasoning, memory, and reflection components inside declared contracts over five rounds of execution feedback, writing a free-form program once, and selecting a library configuration, on DeliveryBench, ALFRED, MiniGrid, RoboTHOR, and DeliveryGym. On DeliveryBench the evolved harness, which compresses the action history into an outcome-aligned ledger and adjudicates between sampled decisions, exceeds both baselines, and its advantage persists on average when the frozen harness is re-executed. On MiniGrid the one-shot programs, which decode the rendered observation into a map and plan on it, exceed the evolved harness by a wide margin on the development suite and on held-out tasks. Reading the generated implementations against their execution records shows that, within the editing scope of this study, the two workflows divided work differently between model and program: evolution refined what the executor saw at each step and kept it in the loop, whereas free-form generation moved state maintenance into code and consulted the executor far less often, at the same recorded success rate on ALFRED and a fraction of the cost. AGENTSPEC makes such divisions of computation inspectable, attributable to specific component changes, and accountable in design, evaluation, and deployment cost.

1 INTRODUCTION

An embodied LLM agent is an executable harness that wraps a base model with perception, memory, reasoning, reflection, and action modules. By deciding what the model observes, which history it receives, and how a proposal becomes an action, the harness matters as much for long-horizon success as the model itself (Ahn et al., 2022; Huang et al., 2022; Wang et al., 2024). Its design is difficult because the useful composition depends on the environment: our component study (Appendix C) shows that memory and reflection can improve or degrade performance depending on their pairing, and configurations rarely transfer between domains such as gridworlds and photorealistic navigation. Harness construction is therefore an empirical search verified by execution.

Coding agents can now automate this search, and systems such as AgentSquare, ADAS, and AFlow already evolve agent designs and programs from execution feedback (Shang et al., 2025; Hu et al., 2025b; Zhang et al., 2025d). Applied to embodied tasks, however, these methods leave the editable design space implicit: it remains unclear which components the agent may modify, what state each component keeps across steps and episodes, and how a performance change is attributed to a specific mechanism. Answering them requires a fixed division of roles: humans define the environments, interfaces, and editing bounds, and the coding agent implements and revises candidates within them.

To make this design space explicit, we present AGENTSPEC, a typed specification and runtime in which each component family has a declared contract and state lifecycle, implementations are registered plugins, and a complete harness is a value that can be serialized, diffed, and replayed. An evolution loop lets a coding agent revise components through execution feedback under an evaluator it

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

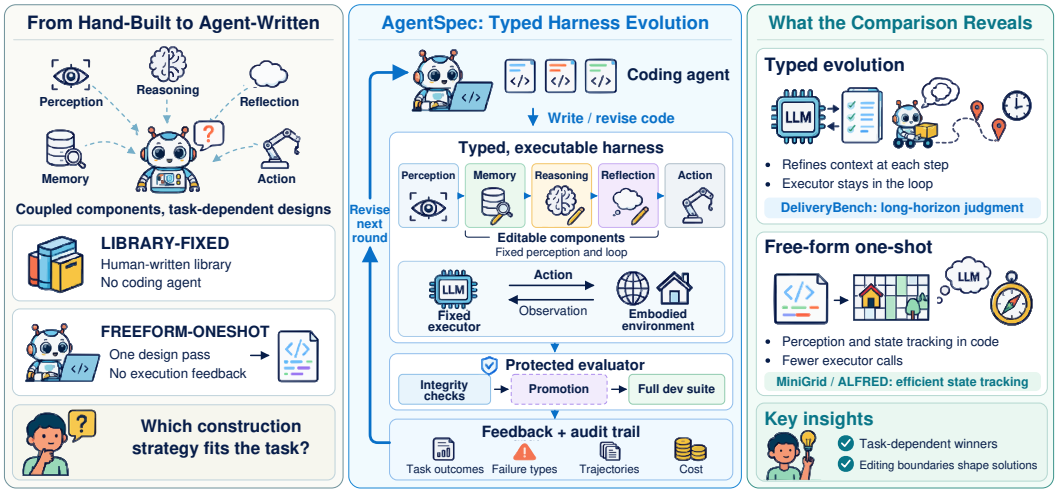


Figure 1: **From hand-built to agent-written harnesses.** Left: we compare LIBRARY-FIXED (a fixed library configuration), FREEFORM-ONESHOT (one standalone harness written by a coding agent without feedback), and AGENTSPEC-EVOLVE. Middle: under AGENTSPEC-EVOLVE a coding agent writes and revises the editable reasoning, memory, and reflection components of a typed, executable harness while perception and the loop stay fixed; the harness drives a fixed executor in the environment, and a protected evaluator runs integrity checks, a gate that selects one candidate per round for complete-suite evaluation, and returns outcomes, failure types, trajectories, and cost for the next round. Right: the designs and results observed in this study. The evolved harnesses refined what the per-step executor sees (best on DeliveryBench); the one-shot programs moved perception and state tracking into code and called the executor rarely (best on MiniGrid; on ALFRED the selected pair reached the same recorded success at a fraction of the cost). The outcome varied by environment; the editing scope shaped the solutions.

cannot bypass, and every candidate is recorded against the change that produced it. With AGENTSPEC we compare three construction workflows under one executor and one designer, which differ in what the coding agent may edit and what it observes: *typed evolution* over five rounds, *free-form one-shot generation* of a standalone harness without feedback, and fixed *library configurations*, evaluated on DeliveryBench (Mao et al., 2025), ALFRED (Shridhar et al., 2020), MiniGrid (Chevalier-Boisvert et al., 2023), RoboTHOR (Deitke et al., 2020), and, at shift level, DeliveryGym (Kang et al., 2026a).

Which workflow wins depends on the environment. On DeliveryBench the evolved harness exceeds both baselines, and the advantage persists on average under re-execution; on MiniGrid the one-shot programs exceed it by a wide margin on the development suite and on held-out tasks; on ALFRED an evolved harness and a one-shot program reach the same recorded success rate with a seventeen-fold difference in executor tokens; DeliveryGym places evolution inside the one-shot range, and RoboTHOR leaves every workflow near zero. Read against their execution records, the generated implementations reveal a consistent division of work between model and program: within this editing scope, evolution refined what the executor saw and kept it in the loop, whereas free-form generation moved state maintenance into code and consulted the executor far less often.

We make three contributions:

- **A typed substrate.** AGENTSPEC makes harness components agent-writable programs with declared contracts, state lifecycles, and execution boundaries, evaluated by a protected evaluator and recorded per candidate (Sections 3 and 4).
- **A controlled comparison.** Three construction workflows on five environments, with campaign records, frozen-harness repeats, and a held-out evaluation (Section 5).
- **An implementation-level account.** What the coding agent changed, how the generated harnesses divide computation between model and program, and what each workflow costs in design, evaluation, and deployment (Section 6).

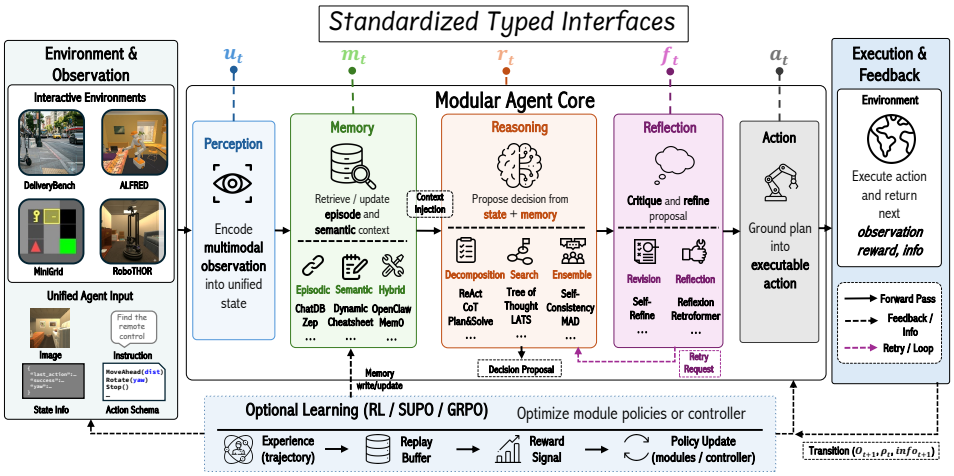


Figure 2: **The inner execution loop.** AGENTSPEC decomposes embodied decision-making into a typed Perception–Memory–Reasoning–Reflection–Action loop; transition feedback updates memory within an episode. The outer evolution loop (Section 4) edits the implementations and configuration of this loop between rounds; the executor's weights are never changed.

2 RELATED WORK

Automated agent and workflow design. AgentSquare (Shang et al., 2025) recombines standardized modules under a target metric, ADAS (Hu et al., 2025b) has a meta-agent write complete agent programs, AFlow (Zhang et al., 2025d) searches graphs of LLM calls from execution feedback, and AgentSwift (Li et al., 2026b), EvoPrompting (Chen et al., 2023), SE-Agent (Lin et al., 2025), and AgentGym (Xi et al., 2025) search designs, evolve code, or train agents. Table 12 (Appendix D) compares the three nearest systems on what is edited, whether component contracts are declared, what feedback the designer sees, whether it can run the benchmark itself, and what the reported number estimates. Building on these systems, we ask what the generated harnesses look like when the editing scope differs, and how their execution records explain their scores.

Modular scaffolds, memory, and reasoning. CoALA and systems such as Generative Agents, MetaGPT, AutoGen, CAMEL, and Voyager (Sumers et al., 2024; Park et al., 2023; Hong et al., 2024; Wu et al., 2024; Li et al., 2023; Wang et al., 2024) compose memory, reasoning, and action modules; the component literature spans chain-of-thought, ReAct, and search-based reasoning (Wei et al., 2022; Yao et al., 2023b;a; Hao et al., 2023), reflection (Madaan et al., 2023; Shinn et al., 2023; Yao et al., 2024), and memory designs from buffers to graph, retrieval-augmented, and procedural stores (Packer et al., 2023; Zhong et al., 2024; Chhikara et al., 2025; Suzgun et al., 2026; Zhang et al., 2026b; Liu et al., 2026; Rasmussen et al., 2025). These works supply the method library (Appendix E); AGENTSPEC adds the contracts that make library and designer-written code interchangeable.

Embodied evaluation. Closed-loop benchmarks (Shridhar et al., 2020; Deitke et al., 2020; Chevalier-Boisvert et al., 2023; Mao et al., 2025; Savva et al., 2019; Fan et al., 2022) expose failures that single-shot tasks hide, and action outcomes let feedback describe behaviour, not solutions.

3 AGENTSPEC: A TYPED HARNESS SUBSTRATE

AGENTSPEC instantiates an embodied agent as a Gym-compatible wrapper around a typed Perception–Memory–Reasoning–Reflection–Action loop (Figure 2). At step t the adapter produces a typed input $u_t = \mathcal{P}(d, o_t)$ from the task d and observation o_t; memory returns context $m_t = \mathcal{M}(u_t, h_{<t})$ and stores the step; reasoning proposes $r_t = \mathcal{R}(u_t, m_t)$ with an extractable action; reflection may revise it to $\hat{r}_t = \mathcal{F}(r_t, u_t, m_t)$ and observe the outcome; and the action module parses, validates, and executes $\hat{r}_t$ under the step budget. Contracts are enforced by base classes and the orchestrator, and every run records each component's inputs, outputs, and token usage, so behaviour can be attributed to the component that produced it. The adapter decides which state fields reach the executor (on MiniGrid, a rendered partial view plus a few flags, never the grid).

What the agent may change. Table 4 (Appendix A.1) fixes, for each component, what it may read and return and what state it keeps: memory persists across steps and is reset at episode start, reasoning is reset per episode and may enrich the experience written to memory, reflection receives the outcome of the executed action through an `observe_result` hook and an episode-end hook, and the action module validates every proposal against the environment's schema. These lifecycle rules are what let an implementation be replaced or authored without touching the others, and they are also where a component can be structurally valid yet semantically wrong, as the ALFRED history of Section 6.3 shows. A candidate is a JSON object naming a reasoning, a memory, and a reflection method with their parameters (Table 5); methods come from the human-written library (5 reasoning $\times$ 6 memory $\times$ 3 reflection backbones) or from *evolved* files that the designer adds to an evolvable directory, each declaring its identity, parent method, entry class, parameters, and a generality statement. Once evaluated, an implementation is frozen; everything outside the evolvable directories (perception, the agent loop, environments, suites, scoring, evaluator, and gate) is protected, a static scan rejects benchmark-specific code, and only the evaluator can run the benchmark (Appendix A). Here the action adapter, perception, and controller stay fixed, so every candidate is compared under one interface and the search runs over components, not architectures; the same contracts can expose these parts too.

4 Agent-Driven Harness Evolution

Roles and loops. A harness h turns the fixed executor B and an environment E into a policy in the program space $\mathcal{H}$ of Table 5; the designer G is a coding-agent session that proposes harnesses and never acts in the environment. The inner loop of Section 3 updates execution state within an episode, whereas the outer loop changes code versions between evaluations. Development feedback comes from the suite D on which the campaign evaluates, and under a budget $\mathcal{B}$ the campaign returns

$$h^* \in \arg\max_{h \in C^{\mathcal{B}}_{\text{valid}}} \widehat{J}_D(h; B), \tag{1}$$

the best of the finitely many candidates actually validated and scored.

Procedure. One persistent designer session drives all rounds of Algorithm 1, starting from the library. Before each round it reads the previous round's *formal feedback* (aggregate metrics, per-task outcomes, failure categories with counts, bounded failure snippets, and per-component executor usage) and the rejected candidates of earlier rounds with their scores; it then submits three versioned candidates with rationales, and a round is refused at no cost if a protected file changed or the generality scan fails. On the gated arms each candidate is scored on a fixed four-task gate subset and only the gate winner is evaluated on the complete suite; on DeliveryBench, whose two tasks are seed replicates of one map, all three candidates run on the complete suite. The winner becomes the parent of the next round, rejected candidates keep their code, and both outcomes return to the designer (Appendix A).

Estimators. Because the complete suite is the *development* suite whose results return to the designer every round, the campaign score measures in-suite optimization and is the maximum over round winners. We therefore report three quantities separately: the round winner's score, the campaign's best-so-far (its development-suite maximum, reported with the round-1 score and round mean), and the score of the frozen best harness re-executed after the campaign (Section 5.2).

Cost. We account cost as $C_{\text{total}}(N) = C_{\text{designer}} + C_{\text{dev-eval}} + N\, C_{\text{deploy}}$: the designer's tokens and time, the cost of every scored evaluation, and the per-episode cost of the locked harness over N deployments.

5 Experimental Evaluation

5.1 Protocol

Environments and suites. DeliveryBench (Mao et al., 2025) is a city-scale delivery simulator scored by mean hourly profit over one-hour episodes; its suite holds 2 seed replicates of one map. ALFRED (Shridhar et al., 2020) provides 10 *valid-unseen* household tasks with per-task caps of $\min(200, 3 \times \text{oracle length})$ steps, MiniGrid (Chevalier-Boisvert et al., 2023) 10 mixed navigation and object tasks seen through first-person partial RGB views under a 100-step cap, and

Algorithm 1 Harness evolution round loop (one campaign; R=5, K=3; blocked candidates cost validation time only)

Require: frozen experiment: environment and suite D, gate subset $G \subset D$ or none, executor B, rounds R, candidates per round K=3; library methods $\mathcal{L}$; designer G_{d} (one persistent session)

$h_{\mathrm{par}} \leftarrow \emptyset$; best $\leftarrow -\infty$; $\mathcal{S} \leftarrow \emptyset$ ▷ $\mathcal{S}$: sealed method files
for $r = 1, \ldots, R$ **do**
  ctx $\leftarrow$ FORMALFEEDBACK(h_{par}), rejected candidates of earlier rounds, trajectories on disk
  $\{(h_i, \mathrm{rationale}_i)\}_{i=1}^{K} \leftarrow G_{\mathrm{d}}(\mathrm{ctx})$ ▷ new methods as new files; evaluated files are immutable
  refuse the round if a protected or sealed file changed, or the generality scan fails ▷ costs no round
  if G exists **then** ▷ gated arms
    $g_i \leftarrow$ EVALUATE(h_i, G) for all i; $i^* \leftarrow \arg\max_i g_i$ (ties: proposal order)
    $s_r \leftarrow$ EVALUATE(h_{i^*}, D)
  else ▷ ungated arm
    $s_i \leftarrow$ EVALUATE(h_i, D) for all i; $i^* \leftarrow \arg\max_i s_i$; $s_r \leftarrow s_{i^*}$
  end if
  $h_{\mathrm{par}} \leftarrow h_{i^*}$; $\mathcal{S} \leftarrow \mathcal{S} \cup \mathrm{files}(h_1, \ldots, h_K)$; best $\leftarrow \max(\mathrm{best}, s_r)$ ▷ the parent may score below its predecessor; all candidates stay in the archive
  record candidates, scores and their kinds, diff, failure categories, resources, rationales
end for
return best and the round trajectory $(s_1, \ldots, s_R)$ ▷ every evaluation goes through the single-use-token evaluator

Table 1: **Recorded complete-suite performance of the executed construction workflows.** Same executor, suites, seeds, step caps, and scoring in every column; one execution per entry. LIBRARY-FIXED: a fixed library configuration without a coding agent (MiniGrid: mean and range over eight configurations; DeliveryBench and RoboTHOR: the library candidate evaluated inside the campaign). FREEFORM-ONESHOT: f02 and f03 share one protocol; f01 is an earlier version. "Best of 5" is the campaign's development-suite maximum (mean over rounds in parentheses); round 1 is already the winner of three screened proposals. †Byte-identical recovery run. ‡Executor faults on most calls (Appendix A.9). §v2 draw under the f02 protocol (Appendix A.9). ¶32-shift validation suite.

Workflow / run	Estimator	DeliveryBench profit ↑	ALFRED SR ↑	MiniGrid SR ↑	RoboTHOR SR ↑	DeliveryGym earnings ↑
LIBRARY-FIXED (h_0, no coding agent)	one configuration	16.31	–	0.37 (0.20–0.50)	0.00	–
FREEFORM-ONESHOT-f01	one draw	12.44	0.50	0.70§	0.00†	7.77 / 4.90 / 7.60
FREEFORM-ONESHOT-f02	one draw	–	0.20‡	0.80	0.10	7.01
FREEFORM-ONESHOT-f03	one draw	–	0.30	0.80	0.00	5.76, 6.57
AGENTSPEC-EVOLVE-f02, round 1	first round	21.77	0.30	0.40	0.00	6.01
AGENTSPEC-EVOLVE-f02, best of 5	max (mean)	**32.07** (22.0)	**0.50** (0.38)	0.40 (0.38)	0.10 (0.02)	6.58¶ (6.08)

RoboTHOR (Deitke et al., 2020) 10 object-navigation episodes with stop-based success; these three report success rate (SR). Suites, seeds, and scoring are frozen and identical across workflows (Appendix F). The executor is Qwen3.5-9B served by vLLM at temperature 1 (0.1 in the DeliveryBench campaign and its repeats); the designer is Claude Opus 5 through Claude Code at high reasoning effort. Each entry of Table 1 is a single execution; Table 2 adds repeats of the frozen harnesses.

Workflows. Under FREEFORM-ONESHOT, a fresh designer session receives the benchmark description, the interface files, and the executor client, writes one standalone agent with neither library access nor feedback, and that agent is evaluated once; f02 and f03 share one protocol and f01 is an earlier version. AGENTSPEC-EVOLVE runs Algorithm 1 under the f02 protocol for 5 rounds of 3 candidates. LIBRARY-FIXED denotes configurations of the AGENTSPEC method library scored under the same protocol with no coding agent: eight MiniGrid configurations run as a calibration set (SR 0.20–0.50) and the library-only candidates the designer submitted inside the campaigns (DeliveryBench CoT+MemoryBank; RoboTHOR ReAct+Base). Because round 1 is already the winner of three screened proposals, the library harness h_0 is reported separately.

Table 2: **Repeated evaluations of the frozen harnesses** (post-hoc; same executor, suites, seeds, and protocol as Table 1; Appendix A.8). "Recorded" is the Table 1 entry, for evolution the best-of-5 maximum. DeliveryBench repeats use the campaign's executor temperature (0.1) for all three harnesses. "Held-out": mean
pm sd over three repeats on ten MiniGrid environment–seed pairs that appeared in no prompt, gate, or feedback.

Harness	Recorded	Repeats	Mean ± sd	Held-out	Calls/ep.
DeliveryBench (mean hourly profit)					
LIBRARY-FIXED (CoT+MemoryBank)	16.31	10.4 / 12.6	11.5 ± 1.6	–	98
FREEFORM-ONESHOT-f01	12.44	17.5 / 13.0 / 10.9	13.8 ± 3.4	–	42
AGENTSPEC-EVOLVE (round-3 winner)	32.07	14.3 / 19.1 / 20.0	17.8 ± 3.1	–	175
MiniGrid (SR)					
LIBRARY-FIXED (ReAct+Base)	0.50	0.40 / 0.20 / 0.30	0.30 ± 0.10	0.37 ± 0.06	69
FREEFORM-ONESHOT-f02	0.80	0.90 / 0.50 / 0.70	0.70 ± 0.20	0.87 ± 0.06	22
FREEFORM-ONESHOT-f03	0.80	0.70 / 0.80 / 0.70	0.73 ± 0.06	0.90 ± 0.10	18
AGENTSPEC-EVOLVE (round-1 winner)	0.40	0.40 / 0.50 / 0.40	0.43 ± 0.06	0.33 ± 0.12	59

5.2 HARNESS-CONSTRUCTION COMPARISON

Table 1 reports every executed arm. First, five rounds of evolution found a better harness than round 1 on DeliveryBench (21.77 to 32.07), ALFRED (0.30 to 0.50), RoboTHOR (0.00 to 0.10), and DeliveryGym (6.01 to 6.58), but not on MiniGrid (0.40), all on the development suite. Second, where a same-protocol library point exists, the evolved harness exceeds it on DeliveryBench (32.07 against 16.31, with even the round mean 22.0 above it) and on RoboTHOR (0.10 against 0.00), and sits inside the library's range on MiniGrid (0.40 against 0.20–0.50 over eight configurations). Third, against one-shot generation the outcome depends on the environment: the evolved harness exceeds the one-shot draw on DeliveryBench, matches the best recorded draw on ALFRED (0.50), matches the best of three on RoboTHOR where every workflow stays near zero, lies inside the one-shot range on DeliveryGym (5.76–7.01), and falls well below the three one-shot draws on MiniGrid (0.70–0.80), whose programs decode the rendered observation into a map (Section 6.3). Each column thus compares complete construction workflows, each with the information its protocol grants (Table 6, Appendix A).

Repeated evaluations. We re-executed the frozen winners and baselines three times on MiniGrid and DeliveryBench (Table 2; Appendix A.8). The mean ordering of the repeats agrees with the campaign comparison: on DeliveryBench the evolved harness averages 17.8 against 13.8 for the one-shot agent and 11.5 for CoT+MemoryBank under a matched executor temperature, and on MiniGrid it averages 0.43 against 0.30 for ReAct+Base while the one-shot agents average 0.70 and 0.73. Re-execution also gives the expected level of the selected DeliveryBench harness, 17.8, which the best-of-five campaign maximum of 32.1 bounds from above; the advantage over both baselines holds at this level. On a held-out MiniGrid suite of ten unseen environment–seed pairs, the one-shot agents keep their level (0.87 and 0.90), whereas the evolved harness (0.33) and ReAct+Base (0.37) remain close, just as the MiniGrid campaign never moved beyond its first round on the development suite; the one-shot advantage therefore carries over to unseen tasks.

DeliveryGym. DeliveryGym (Kang et al., 2026a) extends DeliveryBench to continuous courier shifts; on its 32-shift validation suite six one-shot draws earned 4.9–7.8 and the five evolution rounds 6.01, 5.89, 6.58, 6.02, 5.88 (mean 6.08), so on shift-level delivery the two agent-driven workflows perform on par; the design for the official test shifts is in Appendix A.10.

5.3 EVOLUTION DYNAMICS

Figure 3 shows every round: gains arrive in single rounds (round 3 on DeliveryBench, round 4 on ALFRED) while MiniGrid and RoboTHOR stay flat; candidates within a round were often close on the four-task gate (Appendix A.5); and later rounds vary around the best one within the spread that suites of two and ten tasks allow.

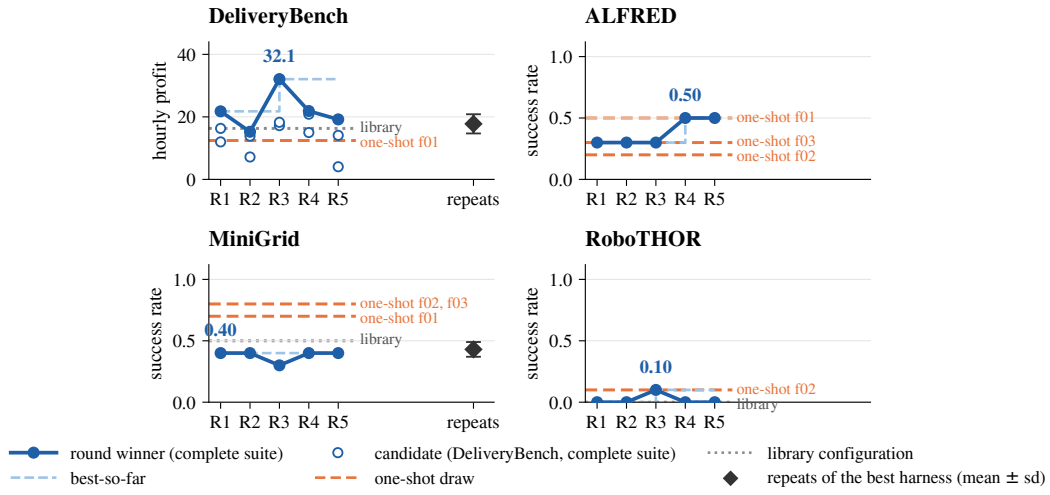


Figure 3: **Round-by-round record of the four evolution arms.** Solid line: each round winner's complete-suite score (the round result); light dashed line: best-so-far (the development-suite maximum reported in Table 1); hollow markers (DeliveryBench): all three candidates' scores (gate scores of the other arms: Appendix A.4); orange dashed: one-shot draws; grey dotted: library configurations under the same protocol; diamond: the frozen best harness re-executed after the campaign, mean ± sd over three repeats (Table 2).

6 Analysis of Harness Design and Evolution

6.1 What Did the Coding Agent Change?

Table 7 (Appendix A) classifies the 35 evolved implementations by what their code does: the designer concentrated on whichever failure mode the feedback exposed, consistent with the failure taxonomy of the component study (Appendix C), writing outcome-aligned history on DeliveryBench and ALFRED, where the library memory restated the previous step's outcome one line late, and commitment and guard mechanisms on MiniGrid and RoboTHOR, where trajectories oscillated and stopped early. Four of the 35 survive into a best observed harness. The evolved methods are 240–430 non-comment lines each, most inheriting from a library class or an earlier evolved version, whereas each one-shot program is a standalone 1,780–2,360-line program with six to eight source files and at most two direct executor call sites (Appendix A.5).

On MiniGrid the designer authored a situated-digest memory, a trace-carrying ReAct variant, and three generations of commit and action guards, yet none moved the score from 0.40: the perception interface it could not edit passes the rendered image with a text field that describes only the cell directly ahead, whereas every one-shot program decoded the image and maintained a map (Section 6.3).

6.2 Where Do the Workflows Differ?

Figure 4 decomposes the scores by task and by executor use. At task level, the library configurations and the evolution rounds on MiniGrid never solve the route-finding tasks (lava crossing, four rooms, the two-wall crossing), in the study or in any repeat of Table 2, whereas the union over the one-shot draws covers every task; on ALFRED evolution solves five tasks and the one-shot draws seven, separated by multi-stage tasks where a program that tracks preconditions beats step-by-step reasoning.

At the level of executor use, the best evolved harnesses consult the executor at nearly every step (77–93 calls per episode on ALFRED, 51–77 on MiniGrid, 174 on DeliveryBench), while the one-shot programs consult it 6–14, 22–24, and 31 times respectively and finish in fewer environment steps. Outcome categories add the diagnosis: on ALFRED the evolved memory cut terminations without success from four episodes to one between rounds 1 and 4 while truncations stayed at three or four, so it repaired wrong-sequence and premature-stop failures but not search failures.

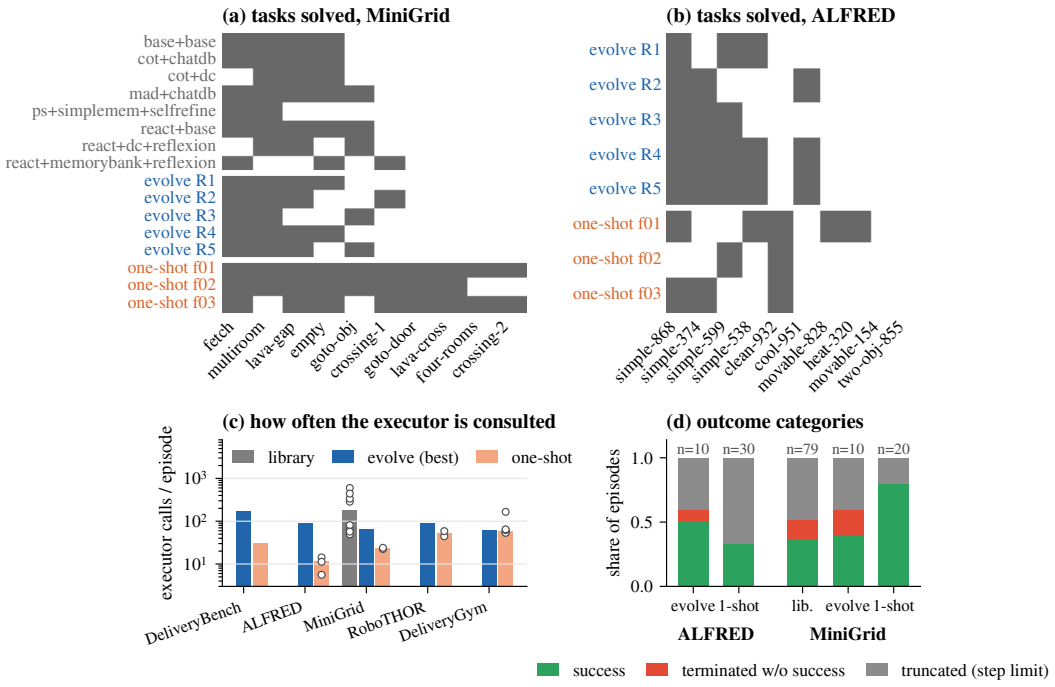


Figure 4: **Task-level and behavioural evidence from the same evaluations.** (a, b) Each row is one complete evaluation (a library configuration, an evolution round winner, or a one-shot draw); dark cells mark tasks solved in that evaluation, and tasks are sorted by how many evaluations solved them. (c) Executor calls per episode (median bar, individual evaluations as points, log scale). (d) Outcome categories pooled over each workflow's evaluations, with the episode count as denominator (terminated: a wrong or premature stop).

Table 3: **Two ALFRED harnesses with the same recorded success rate.** Recorded evaluations of the round-4 winner and one-shot draw f01 (Appendix B), not the repeat statistics of Table 2.

	Evolved harness (ReAct + `OutcomeTraceMemV1` + `ActionCommitV1`)	**One-shot program** (f01)
Main design	History records each step's own outcome; model-driven step loop kept	Geometry, occupancy map, and precondition repair in code
Executor's role	Chooses the action at every step	Interprets the instruction; guesses object locations
Recorded SR	0.50	0.50
Executor calls / episode	90	14
Executor tokens / episode	0.52M	0.03M

6.3 How Do the Generated Harnesses Divide Computation?

On ALFRED (Table 3) the evolved harness kept the library's model-driven loop and repaired its history so that each step's line carries that step's outcome, while the one-shot program keeps geometry, mapping, and precondition repair in code and consults the executor only for instruction interpretation and object-location guesses; both reach 0.50 on the recorded evaluation at a six-fold difference in calls. On DeliveryBench the evolved harness compresses the action history into a one-line-per-step ledger and asks the executor to adjudicate between sampled chains of thought only when they disagree (174 calls per episode against 31 for the one-shot dispatch program that encodes fare, rating, and clock rules in code); its advantage is consistent with this design and persists on average in repeated execution. On MiniGrid the evolved harness changed what the executor was told but not what it could see, whereas all three one-shot programs inverted the renderer into symbolic cells and planned on a persistent map (Appendix B audits all three cases on the raw records).

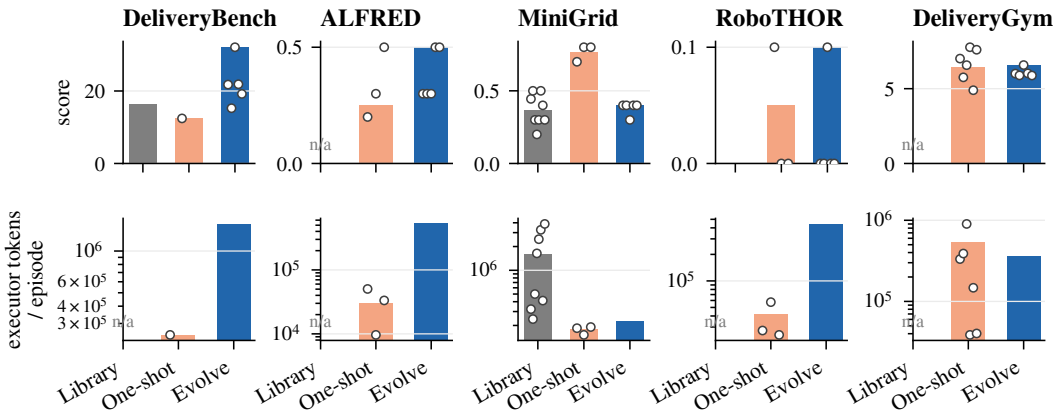


Figure 5: **Score and deployment cost per construction workflow.** Top: complete-suite score; bottom: executor tokens per episode of the final harness (log). Bars: mean over library configurations (MiniGrid: eight; elsewhere the single library candidate), mean over same-protocol one-shot draws, and the evolution campaign's best harness; points: the individual runs behind each bar. Design and evaluation cost: Figure 8, Appendix A.

Three observations follow. First, similar recorded scores can hide different divisions of computation (the two ALFRED programs of Table 3). Second, interface compatibility is not state semantics: the library ALFRED memory satisfied its contract while misaligning actions and outcomes by one step, the repair was a change to the state-recording logic that the typed lifecycle located in one component, and the execution record showed it as fewer terminations without success (Section 6.2); the substrate makes such defects findable and attributable. Third, the editing scope determines which improvements can be expressed: the map-building programs that decided MiniGrid lie outside the evolvable components of this study. More broadly, the value of keeping the executor in the loop depends on the task and the program, as DeliveryBench, ALFRED, and DeliveryGym together show.

6.4 What Does Each Workflow Cost?

Figure 5 and Table 9 (Appendix A) separate three cost terms. Designer cost is modest (about two one-shot draws' worth); evaluation cost is where iteration becomes expensive (23–70M executor tokens and 8–21 hours per environment against at most 1.5M tokens and 1.4 hours per draw); deployment cost separates the workflows most sharply (0.22–1.6M executor tokens per episode for evolved harnesses, inside the library's band, against 0.01–0.39M for one-shot programs). Under $C_{\text{total}}(N)$ an evolved harness therefore pays for its accuracy per episode, a price justified where in-loop judgement raises the score, as on DeliveryBench, and not where a program can carry the state, as on ALFRED.

7 Conclusion

AgentSpec turns an embodied-agent harness into a typed, executable program that a coding agent can modify, compose, run, and trace, with every candidate recorded against the change that produced it. Under one executor and one designer, the two agent-driven workflows built different agents within this editing scope: evolution refined what the executor saw at each step and kept it in the loop, which won on DeliveryBench and persisted on average under repeated execution; free-form generation moved perception and state maintenance into code and consulted the executor far less often, which won on MiniGrid, generalized to held-out tasks, and matched the evolved harness on ALFRED at a fraction of the cost. Evaluating agent-written harnesses therefore requires stating what the agent was allowed to change and how the resulting program divides work between model and code, not only its score; AgentSpec makes both inspectable. Because perception and the loop order are components under the same contracts, the runtime extends directly to campaigns in which the coding agent may also rewrite what the executor sees, the change that decided MiniGrid.

AI USE STATEMENT

We used generative AI tools to check and polish the writing of this manuscript (grammar, clarity, and consistency of terminology); all edits were reviewed by the authors. We did not use generative AI tools to develop the conceptual framework, propose or refine hypotheses, design the experiments, or interpret the results, and the remaining tasks with required disclosure (synthetic data, mathematical claims and proofs, translation, dataset cleaning, qualitative data analysis) are not applicable. Coding-agent and language-model use that forms part of the experimental method (the designer and executor of Sections 4 and 5) is described in the paper and reported as an experimental outcome. The authors take responsibility for the final content of this work.

ETHICS STATEMENT

This work studies LLM agents in simulated embodied environments (DeliveryBench, DeliveryGym, ALFRED, MiniGrid, and RoboTHOR). It involves no human subjects, no personal data, and no physical robots, and all benchmarks are publicly available and used under their respective licenses. Harness code written by coding agents is executed only against these simulated benchmark environments and does not interact with real-world systems. We do not foresee direct negative societal impacts from this work; as with other research on autonomous agents, more capable and more autonomous scaffolds should be deployed with appropriate human oversight. The authors have read and adhere to the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

Sections 3 and 4 specify the AGENTSPEC contracts, the agent-writable design space, and the evolution procedure as executed (Algorithm 1); Section 5.1 and Appendix F state the executor, designer, suites, step budgets, information flow, and evaluation budgets; Appendix A gives the designer's task and write scope, the gate and feedback contents, every round score and every DeliveryBench candidate score, the selected configurations, the status of every run, the intervention log, the repeated-evaluation protocol, and the DeliveryGym protocol; the human-written library is in Appendix E and the component study in Appendix C. The campaign artifacts (candidate specifications, plugin sources, validation reports, sharded evaluations, result histories, and designer notebooks) and the figure scripts will be released with the code so that both the inner execution and the outer selection can be replayed.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. AriGraph: Learning knowledge graph world models with episodic memory for LLM agents. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pp. 12–20, 2025.

Simon A. Aytes, Jinheon Baek, and Sung Ju Hwang. Sketch-of-Thought: Efficient LLM reasoning with adaptive cognitive-inspired sketching. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 24296–24320, 2025.

Debangshu Banerjee, Tarun Suresh, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. CRANE: Reasoning with constrained LLM generation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 2836–2857, 2025.

Massimo Bini, Ondrej Bohdal, Umberto Michieli, Zeynep Akata, Mete Ozay, and Taha Ceritli. MemLoRA: Distilling expert adapters for on-device memory systems. *arXiv preprint arXiv:2512.04763*, 2025.

Jiaqi Cao, Jiarui Wang, Rubin Wei, Qipeng Guo, Kai Chen, Bowen Zhou, and Zhouhan Lin. Memory decoder: A pretrained, plug-and-play memory for large language models. In *Advances in Neural Information Processing Systems*, 2025.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16803–16822, 2026.

Maitreyi Chatterjee and Devansh Agarwal. Semantic anchoring in agentic memory: Leveraging linguistic structures for persistent conversational context. *arXiv preprint arXiv:2508.12630*, 2025.

Angelica Chen, David Dohan, and David So. EvoPrompting: Language models for code-level neural architecture search. *Advances in neural information processing systems*, 36:7787–7817, 2023.

Weishu Chen, Jinyi Tang, Zhouhui Hou, Shihao Han, Mingjie Zhan, Zhiyuan Huang, Delong Liu, Jiawei Guo, Zhicheng Zhao, and Fei Su. MOOM: Maintenance, organization and optimization of memory in ultra-long role-playing dialogues. *arXiv preprint arXiv:2509.11860*, 2025.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Xuefei Cao, Yuanhao Xiong, Haibo Tong, Huaxiu Yao, Hengduo Li, Jiacheng Zhu, Xian Li, Dawn Song, Bo Li, Jason Weston, and Dat Huynh. Scaling agent learning via experience synthesis. In *The Fourteenth International Conference on Learning Representations*, 2026.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and J K Terry. Minigrid & Miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Advances in Neural Information Processing Systems*, 36:73383–73394, 2023.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. In *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI 2025)*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pp. 2993–3000, 2025.

Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford, Luca Weihs, Mark Yatskar, and Ali Farhadi. RoboTHOR: An open simulation-to-real embodied AI platform. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3164–3174, 2020.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first international conference on machine learning*, 2024.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. MineDojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35: 18343–18362, 2022.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. LightMem: Lightweight and efficient memory-augmented generation. In *The Fourteenth International Conference on Learning Representations*, 2026a.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, 2026b.

Saman Forouzandeh, Wei Peng, Parham Moradi, Xinghuo Yu, and Mahdi Jalili. Learning hierarchical procedural memory for LLM agents through bayesian selection and contrastive refinement. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2026.

Kuofeng Gao, Huanqia Cai, Qingyao Shuai, Dihong Gong, and Zhifeng Li. Embedding self-correction as an inherent ability in large language models for enhanced mathematical reasoning. *arXiv preprint arXiv:2410.10735*, 2024a.

Zitian Gao, Boye Niu, Xuzheng He, Haotian Xu, Hongzhang Liu, Aiwei Liu, Xuming Hu, and Lijie Wen. Interpretable contrastive monte carlo tree search reasoning. *arXiv preprint arXiv:2410.01707*, 2024b.

Dongge Han, Camille Couturier, Daniel Madrigal Diaz, Xuchao Zhang, Victor Rühle, and Saravan Rajmohan. LEGOMem: Modular procedural memory for multi-agent LLM systems for workflow automation. *arXiv preprint arXiv:2510.04851*, 2025.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, 2023.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*, 2025.

Alexander Havrilla, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, and Roberta Raileanu. GLoRe: When, where, and how to improve LLM reasoning via global and local refinements. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 17719–17733, 2024.

Matthew Ho, Chen Si, Zhaoxiang Feng, Fangxu Yu, Yichi Yang, Zhijian Liu, Zhiting Hu, and Lianhui Qin. ArcMemo: Abstract reasoning composition with lifelong LLM memory. *arXiv preprint arXiv:2509.04439*, 2025.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024.

Yining Hong, Huang Huang, Manling Li, Li Fei-Fei, Leonidas Guibas, Jiajun Wu, and Yejin Choi. Learning from trials and errors: Reflective test-time planning for embodied LLMs. *arXiv preprint arXiv:2602.21198*, 2026.

Chenxu Hu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. ChatDB: Augmenting LLMs with databases as their symbolic memory. *arXiv preprint arXiv:2306.03901*, 2023.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. HiAgent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32779–32798, 2025a.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Yuyang Hu, Jiongnan Liu, Jiejun Tan, Yutao Zhu, and Zhicheng Dou. Memory matters more: Event-centric memory as a logic map for agent searching and reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 22389–22407, 2026.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782, 2022.

Dongming Jiang, Yi Li, Guanpeng Li, and Bingzhe Li. MAGMA: A multi-graph based agentic memory architecture for AI agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 36848–36865, 2026.

Haoqiang Kang, Enna Sachdeva, Piyush Gupta, Sangjae Bae, and Kwonjoon Lee. GFlowVLM: Enhancing multi-step reasoning in vision-language models with generative flow networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3815–3825, 2025.

Haoqiang Kang, Yiming Zhang, Yiyang Guo, Chuying Li, Jianzhi Shen, Tianruo Rose Xu, Xiaokang Ye, and Lianhui Qin. DeliveryGym: An RL environment for long-horizon embodied agent planning with adaptive curriculum. *arXiv preprint arXiv:2609.19801*, 2026a.

Haoqiang Kang, Yizhe Zhang, Nikki Lijing Kuang, Yi-An Ma, and Lianhui Qin. LaDi-RL: Latent diffusion reasoning prevents entropy collapse in reinforcement learning. *arXiv preprint arXiv:2602.01705*, 2026b.

Mehran Kazemi, Najoung Kim, Deepti Bhatia, Xin Xu, and Deepak Ramachandran. LAMBADA: Backward chaining for automated reasoning in natural language. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6547–6568, 2023.

Jack Lanchantin, Shubham Toshniwal, Jason Weston, Arthur Szlam, and Sainbayar Sukhbaatar. Learning to reason and memorize with self-notes. *Advances in Neural Information Processing Systems*, 36:11891–11911, 2023.

LangChain AI. LangMem. https://github.com/langchain-ai/langmem, 2025. Software library for long-term agent memory; documentation at https://langchain-ai.github.io/langmem/.

Chris Latimer, Nicoló Boschi, Andrew Neeser, Chris Bartholomew, Gaurav Srivastava, Xuan Wang, and Naren Ramakrishnan. Hindsight is 20/20: Building agent memory that retains, recalls, and reflects. *arXiv preprint arXiv:2512.12818*, 2025.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008, 2023.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, Weiyu Liu, Percy Liang, Li Fei-Fei, Jiayuan Mao, and Jiajun Wu. Embodied agent interface: Benchmarking LLMs for embodied decision making. *Advances in Neural Information Processing Systems*, 37:100428–100534, 2024a.

Rui Li, Zeyu Zhang, Xiaohe Bo, Zihang Tian, Xu Chen, Quanyu Dai, Zhenhua Dong, and Ruiming Tang. CAM: A constructivist view of agentic memory for LLM-Based reading comprehension. In *Advances in Neural Information Processing Systems*, 2025a.

Shilong Li, Yancheng He, Hangyu Guo, Xingyuan Bu, Ge Bai, Jie Liu, Jiaheng Liu, Xingwei Qu, Yangguang Li, Wanli Ouyang, Wenbo Su, and Bo Zheng. GraphReader: Building graph-based agent to enhance long-context abilities of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 12758–12786, 2024b.

Shuozhe Li, Vaishnav Tadiparthi, Kwonjoon Lee, Nakul Agarwal, Hossein Nourkhiz Mahjoub, Ehsan Moradi Pari, Lizhang Chen, Amy Zhang, and Liu Leqi. Learning robust reasoning through guided adversarial self-play. *arXiv preprint arXiv:2602.00173*, 2026a.

Yu Li, Lehui Li, Zhihao Wu, Qingmin Liao, Jianye Hao, Kun Shao, and Fengli Xu. AgentSwift: Efficient LLM agent design via value-guided hierarchical search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 31843–31851, 2026b.

Yuan Li, Lichao Sun, and Yixuan Zhang. MetaAgents: Large language model based agents for decision-making on teaming. *Proceedings of the ACM on Human-Computer Interaction*, 9(2): 1–27, 2025b.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2024.

Jiaye Lin, Yifu Guo, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, Mingguang Chen, Hongzhang Liu, Ronghao Chen, Yangfan He, Daxin Jiang, Binxing Jiao, Chen Hu, and Huacan Wang. SE-Agent: Self-evolution trajectory optimization in multi-step reasoning with LLM-based agents. *arXiv preprint arXiv:2508.02085*, 2025.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. SimpleMem: Efficient lifelong memory for LLM agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Junming Liu, Yifei Sun, Weihua Cheng, Haodong Lei, Yirong Chen, Licheng Wen, Xuemeng Yang, Daocheng Fu, Pinlong Cai, Nianchen Deng, Yi Yu, Shuyue Hu, Botian Shi, and Ding Wang. MemVerse: Multimodal memory for lifelong learning agents. *arXiv preprint arXiv:2512.03627*, 2025a.

Lei Liu, Xiaoyan Yang, Yue Shen, Binbin Hu, Zhiqiang Zhang, Jinjie Gu, and Guannan Zhang. Think-in-Memory: Recalling and post-thinking enable LLMs with long-term memory. *arXiv preprint arXiv:2311.08719*, 2023a.

Ruibo Liu, Jason Wei, Shixiang Shane Gu, Te-Yen Wu, Soroush Vosoughi, Claire Cui, Denny Zhou, and Andrew M. Dai. Mind's eye: Grounded language model reasoning through simulation. In *The Eleventh International Conference on Learning Representations*, 2023b.

Weijie Liu, Zecheng Tang, Juntao Li, Kehai Chen, and Min Zhang. MemLong: Memory-augmented retrieval for long text modeling. *arXiv preprint arXiv:2408.16967*, 2024.

Yitao Liu, Chenglei Si, Karthik R Narasimhan, and Shunyu Yao. Contextual experience replay for self-improvement of language agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14179–14198, 2025b.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2023.

Lingjun Mao, Jiawei Ren, Kun Zhou, Jixuan Chen, Ziqiao Ma, and Lianhui Qin. DeliveryBench: Can agents earn profit in real world? *arXiv preprint arXiv:2512.19234*, 2025.

Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. RET-LLM: Towards a general read-write memory for large language models. *arXiv preprint arXiv:2305.14322*, 2023.

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip HS Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. MALT: Improving reasoning with multi-agent LLM training. In *Second Conference on Language Modeling*, 2025.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-Thought: Prompting LLMs for efficient parallel generation. In *The Twelfth International Conference on Learning Representations*, 2024.

OpenClaw. OpenClaw. https://github.com/openclaw/openclaw, 2026.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. REFINER: Reasoning feedback on intermediate representations. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1100–1126, 2024.

Hongjin Qian, Zheng Liu, Peitian Zhang, Kelong Mao, Defu Lian, Zhicheng Dou, and Tiejun Huang. MemoRAG: Boosting long context processing with global memory-enhanced retrieval augmentation. In *Proceedings of the ACM on Web Conference 2025*, pp. 2366–2377, 2025.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied AI research. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9339–9347, 2019.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. AgentSquare: Automatic LLM agent search in modular design space. In *The Thirteenth International Conference on Learning Representations*, 2025.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10740–10749, 2020.

Zayne Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. To CoT or not to CoT? chain-of-thought helps mainly on math and symbolic reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, Ge Zhang, Jiaheng Liu, Xingyao Wang, Sirui Hong, Chenglin Wu, Hao Cheng, Chi Wang, and Wangchunshu Zhou. Agent KB: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*, 2025.

Ao Tian, Yunfeng Lu, Xinxin Fan, Changhao Wang, Lanzhi Zhou, Yeyao Zhang, and Yanfang Liu. RGMem: Renormalization group-inspired memory evolution for language agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Bing Wang, Xinnian Liang, Jian Yang, Hui Huang, Zhenhe Wu, ShuangZhi Wu, Zejun Ma, and Zhoujun Li. SCM: Enhancing large language model with self-controlled memory framework. In *International Conference on Database Systems for Advanced Applications*, pp. 188–203. Springer, 2025a.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

Jiaan Wang, Fandong Meng, Yunlong Liang, and Jie Zhou. DRT: Deep reasoning translation via long chain-of-thought. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6770–6782, 2025b.

Juyuan Wang, Rongchen Zhao, Wei Wei, Yufeng Wang, Mo Yu, Jie Zhou, Jin Xu, and Liyan Xu. ComoRAG: A cognitive-inspired memory-organized RAG for stateful long narrative reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33557–33565, 2026.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2609–2634, 2023a.

Xiaoqiang Wang, Suyuchen Wang, Yun Zhu, and Bang Liu. R^3Mem: Bridging memory retention and retrieval via reversible compression. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4541–4557, 2025c.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023b.

Yu Wang and Xi Chen. MIRIX: Multi-agent memory system for LLM-Based agents. *arXiv preprint arXiv:2507.07957*, 2025.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911, 2025d.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-Thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-Memory: Benchmarking LLM agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations*, 2023.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *First Conference on Language Modeling*, 2024.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. EvolveR: Self-evolving LLM agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. AgentGym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27914–27961, 2025.

Violet Xiang, Charlie Snell, Kanishk Gandhi, Alon Albalak, Anikait Singh, Chase Blagden, Duy Phung, Rafael Rafailov, Nathan Lile, Dakota Mahan, Louis Castricato, Jan-Philipp Fränken, Nick Haber, and Chelsea Finn. Towards system 2 reasoning in LLMs: Learning how to think with meta chain-of-thought. *arXiv preprint arXiv:2501.04682*, 2025.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. *Advances in neural information processing systems*, 37:119638–119661, 2024.

Yunzhong Xiao, Yangmin Li, Hewei Wang, Yunlong Tang, and Zora Zhiruo Wang. ToolMem: Enhancing multimodal agents with learnable tool capability memory. *arXiv preprint arXiv:2510.06664*, 2025.

Siheng Xiong, Ali Payani, Yuan Yang, and Faramarz Fekri. Deliberate reasoning in language models as structure-aware planning with an accurate world model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31900–31931, 2025a.

Wei Xiong, Hanning Zhang, Chenlu Ye, Lichang Chen, Nan Jiang, and Tong Zhang. Self-rewarding correction for mathematical reasoning. *arXiv preprint arXiv:2502.19613*, 2025b.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 17577–17604, 2025.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, 2026.

Hongkang Yang, Zehao Lin, Wenjin Wang, Hao Wu, Zhiyu Li, Bo Tang, Wenqiang Wei, Jinbo Wang, Zeyun Tang, Shichao Song, Chenyang Xi, Yu Yu, Kai Chen, Feiyu Xiong, Linpeng Tang, and Weinan E. Memory^3: Language modeling with explicit memory. *arXiv preprint arXiv:2407.01178*, 2024a.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024b.

Sen Yang, Yafu Li, Wai Lam, and Yu Cheng. Multi-LLM collaborative search for complex problem solving. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 42599–42614, 2026a.

Wei Yang, Jiacheng Pang, Shixuan Li, Paul Bogdan, Stephen Tu, and Jesse Thomason. Maestro: Learning to collaborate via conditional listwise policy optimization for multi-agent LLMs. *arXiv preprint arXiv:2511.06134*, 2025.

Wei Yang, Shixuan Li, Heng Ping, Peiyu Zhang, Paul Bogdan, and Jesse Thomason. Auditing multi-agent LLM reasoning trees outperforms majority vote and LLM-as-Judge. *arXiv preprint arXiv:2602.09341*, 2026b.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2023b.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. Retroformer: Retrospective large language agents with policy gradient optimization. In *The Twelfth International Conference on Learning Representations*, 2024.

Woongyeong Yeo, Kangsan Kim, Jaehong Yoon, and Sung Ju Hwang. WorldMM: Dynamic multimodal memory agent for long video reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Fangxu Yu, Lai Jiang, Haoqiang Kang, Shibo Hao, and Lianhui Qin. Flow of reasoning: Training LLMs for divergent reasoning with minimal examples. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 73115–73145, 2025.

Guibin Zhang, Muxin Fu, Kun Wang, Guancheng Wan, Miao Yu, and Shuicheng Yan. G-Memory: Tracing hierarchical memory for multi-agent systems. In *Advances in Neural Information Processing Systems*, volume 38, pp. 12988–13018, 2025a.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. MemEvolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*, 2025b.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. MemSkill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026a.

Hongxin Zhang, Zheyuan Zhang, Zeyuan Wang, Zunzhe Zhang, Lixing Fang, Qinhong Zhou, and Chuang Gan. Ella: Embodied social agents with lifelong memory. *arXiv preprint arXiv:2506.24019*, 2025c.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025d.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *The Fourteenth International Conference on Learning Representations*, 2026b.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 19724–19731, 2024.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160, 2024a.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. Self-Discover: Large language models self-compose reasoning structures. *Advances in Neural Information Processing Systems*, 37: 126032–126058, 2024b.

APPENDIX

APPENDIX CONTENTS

A Designer Contract, Editing Protocol, and Evolution Traces **19**

A.1 Typed Contracts and Agent-Writable Design Space 20

A.2 Designer Task and Write Scope 21

A.3 Gate, Feedback, and Integrity 21

A.4 Round Records 21

A.5 Log-Only Analyses: Gate Diagnostics, Code, and Designer Effort 22

A.6 Design Cost . . . 23
A.7 Pilot Campaign with Behavioural Digests and a Held-Out Suite . . . 23
A.8 Repeated Evaluations of the Frozen Harnesses . . . 24
A.9 One-Shot Draws, Library Calibration, and DeliveryGym Records . . . 26
A.10 DeliveryGym Protocol and External References . . . 26
A.11 Human Intervention Log . . . 26

B Case Studies: A Mechanism-Level Audit **27**
B.1 ALFRED: Interface Compatibility Is Not State Semantics . . . 27
B.2 DeliveryBench: A Compressed Ledger and Adjudication on Disagreement . . . 28
B.3 MiniGrid: What the Executor Could See . . . 29
B.4 The ALFRED Lineage End to End . . . 29

C Controlled Component Studies **30**
C.1 Overall Performance Across Tasks . . . 30
C.2 How Modules Interact . . . 32
C.3 Efficiency–Performance Trade-off . . . 32
C.4 Detailed Efficiency Analysis . . . 33
C.5 Case Study and Error Analysis . . . 35

D Extended Related Work **36**

E Module Implementations **37**
E.1 Reasoning Methods . . . 37
E.2 Memory Methods . . . 37
E.3 Reflection Methods . . . 39

F Detailed Evaluation Settings **40**
F.1 DeliveryBench . . . 40
F.2 Minigrid . . . 41
F.3 Alfred . . . 41
F.4 Suites of the Main Workflow Study . . . 42
F.5 Search and Held-out Suites of the Pilot Campaign . . . 42

G Scope of the Study **42**

A Designer Contract, Editing Protocol, and Evolution Traces

Source tag: final controlled evaluation (f02 campaigns), except where a subsection is tagged pilot. Change categories used in Table 7: state representation or digest; history compression and outcome-aligned trace; task-progress and budget tracking; action commitment and plan persistence; candidate-

action comparison; action validation and failure-recovery guards. Each evolved file was assigned from its code, not its name; the assignment table is released with the artifacts.

A.1 Typed Contracts and Agent-Writable Design Space

Table 4 lists the contract of each component family as enforced by the base classes and the orchestrator; Table 5 lists what the designer may change and how each boundary is enforced.

Table 4: **Typed runtime contracts.** Each component reads and returns typed objects; the loop order and the environment side are fixed, so one implementation can be replaced or authored without touching the others.

Component	Input → output	State and behaviour fixed by the contract
Perception (adapter)	raw observation, task, info → `UnifiedAgentInput`: environment meta, task spec, structured state, visual and text inputs, action schema	one adapter per environment, reset per episode; it decides which state fields are exposed, so hidden simulator state never reaches the executor; environments extend prompts only through declared addendum fields
Memory	current input, k → retrieved items; per-step experience → store	episode-start, reset, and clear lifecycle; a renderer turns an item into prompt text; retrieval depth is an option
Reasoning	input, retrieved memory, profile → dictionary with an extractable action and rationale	one executor-call pattern per method; may enrich the experience written to memory; reset per episode
Reflection	reasoning output, input, memory → refined output	may receive the outcome of the executed action (`observe_result`) and an episode-end hook
Action	refined output → normalized, schema-checked action → environment call	parse with a fallback chain (`action`, then `actions[0]`); validation against the action schema; step budget and termination enforced by the environment runner

Table 5: **Agent-writable design space of the final study.** Evolvable paths are append-only; protected paths are hash-verified before every round; the benchmark can only be executed by the formal evaluator.

Element	Status	Where it lives / how it is enforced
Method selection (reasoning, memory, reflection)	editable	candidate JSON; names must resolve in the registry
Approved hyperparameters and declared method parameters	editable	candidate JSON; values outside the approved domain are rejected
New reasoning, memory, reflection implementations	editable, append-only	one new file with a `METHOD_SPEC` (inherits, entry, params, generality) under `modules/{reasoning,memory,reflection}`; evaluated files are hash-sealed
Perception, agent loop, runner, environments, suites, seeds, scoring	protected	hash-pinned; any change refuses the round
Evaluator, gate policy, budget ledger, this scope file	protected	hash-pinned; the designer never grades its own round
Benchmark execution	evaluator only	runner refuses to start without a single-use authorization token
Benchmark-specific literals (environment imports, task or suite ids, seed branches)	refused	static generality scan before evaluation; hard cases recorded for review
Human intervention	infrastructure only	round budget 10→5 (during the study, uniform, all arms), render-GPU migration for the THOR arms from round 4; listed in Appendix A

A.2 Designer Task and Write Scope

The designer (a Claude Code session pinned to Claude Opus 5, high effort, one persistent session per arm) receives a task document stating that it evolves general reasoning, memory, and reflection mechanisms rather than a benchmark solver; that it may read anything in the repository including the previous round's trajectories; that it may write only new method files under the evolvable directories and its own candidate and notebook files; that it may not run the benchmark; and that every new method must declare a generality statement. It may verify candidates without the benchmark: compilation, import, interface checks, unit tests with fake clients, and a no-budget identity check. Each round it submits exactly three candidates with rationales and runs one blocking round command. A round is refused, at no cost, if a protected or sealed file changed, the candidate is malformed, or the generality scan finds an environment import, a task or suite identifier, or a seed-keyed branch.

A.3 Gate, Feedback, and Integrity

On the gated arms the gate subset is the first four tasks of the suite in declared order (MiniGrid: empty-room, fetch, lava-crossing, multi-room; ALFRED: simple placement, two-object placement, heat-then-place, movable receptacle; RoboTHOR: two tasks from each scene group). The formal feedback of a round contains the aggregate metrics, per-task outcomes, failure categories (success, terminated without reward, truncated at the step limit, error, timeout) with counts, up to a few representative failure snippets, the candidate identity, and exact executor token usage per component. Every evaluation, gate or full, requires a single-use authorization token issued by the evaluator; runs are sharded over the fixed worker count and merged; all 20 full-suite evaluations and 43 gate evaluations of the four arms completed with every task present. Protocol notes: the round budget was set to 5 for all arms uniformly while the campaigns were in progress (from an initial allocation of 10), without re-running any completed round; the ALFRED and RoboTHOR render GPU was migrated from round 4 (endpoint unchanged). The DeliveryBench arm's scores are read from the per-round history, which is complete.

A.4 Round Records

Table 6: **Information and evaluation budget of each executed workflow.** The evolution workflow observes results between rounds; the one-shot workflows observe nothing after generation. The substrate controls AgentSpec-OneShot and matched-budget Freeform-Evolve are defined for completeness and lie outside this study.

Workflow	Proposals	Rounds	Feedback to the designer	Reported outcome
Library-Fixed (h_0)	1	0	none	one full-suite evaluation
Freeform-OneShot-f01 / -f02 / -f03	1	0	none after generation	one full-suite evaluation each
AgentSpec-Evolve-f02	15	5	gate and full-suite results between rounds	best observed full-suite score
AgentSpec-OneShot, Freeform-Evolve	defined; outside this study			

Table 7: **What the designer changed, by functional category and environment** (source: the sealed method files and candidate diffs of the four f02 campaigns). Each cell counts functional-category labels over the evolved method files the designer submitted (a file that implements two mechanisms carries two labels, so columns can exceed the file count); ✓ marks the category present in the campaign's best observed harness. Categories are assigned from the code, not from method names (Appendix A).

Change category	DeliveryBench	ALFRED	MiniGrid	RoboTHOR
State representation / digest of the situation	–	1	2✓	2
History compression, outcome-aligned trace	3✓	3✓	–	1
Task-progress and budget tracking	1	1	–	2
Action commitment, plan persistence	–	1✓	3	3✓
Candidate-action comparison (sampling, selection)	3✓	–	–	–
Action validation, failure recovery guards	1	2	4	4
Evolved method files / rounds with new code	7 / 5	7 / 5	10 / 5	11 / 5

Static evidence and interventional design (Section 6.1). The component study (Appendix C) supplies static evidence: planning-oriented reasoners gain little from memories that return raw observations but substantially from abstract ones, and reflection helps weak pairs most; it used other backbones, so it is context rather than a matched baseline. The interventional counterpart is the substrate control AGENTSPEC-ONESHOT of Table 6, a paired-swap design that evaluates $\{R_0M_0, R^*M_0, R_0M^*, R^*M^*\}$ on each evolved harness, where R_0, M_0 are the library reasoning and memory and R^*, M^* the evolved ones, reports $I_{RM} = J(R^*, M^*) - J(R^*, M_0) - J(R_0, M^*) + J(R_0, M_0)$ with paired intervals over repeated executions, and removes one mechanism at a time (the action ledger, the commitment check, the outcome-aligned update) from the best observed harness. The executed ALFRED lineage already contains one such cell: the round-3 configuration with library memory and commit reflection scored 0.30 in three evaluations, and adding the outcome-aligned memory raised it to 0.50, the campaign's best (Appendix B.1).

Table 8 lists every round winner; Figure 6 adds the per-candidate gate-subset scores. Gate and complete-suite scores differ in both directions (MiniGrid gate 0.75 against complete 0.40; ALFRED gate 0.00 against complete 0.50) because the four gate tasks differ in difficulty from the rest of the suite; the gate's role is to select one candidate per round for complete-suite evaluation, and Appendix A.5 quantifies how much discrimination it provided.

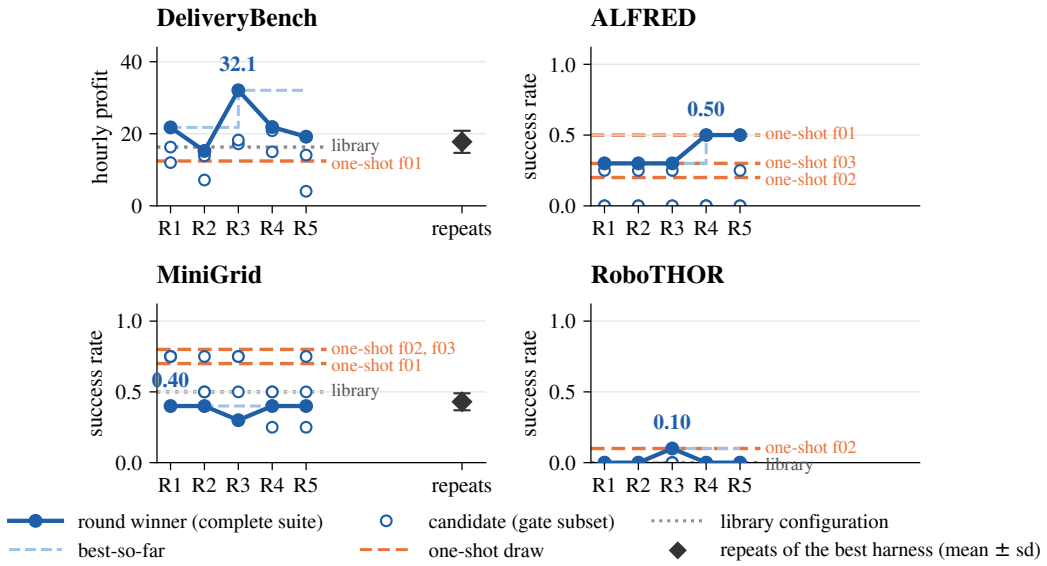


Figure 6: Round record with the three candidates' gate-subset scores of each round (hollow markers) added to Figure 3.

Library-only candidates evaluated inside the campaigns: DeliveryBench CoT+MemoryBank+None 16.3 (full suite, round 1); ALFRED ReAct+Base+None 0.00 (gate only, round 1); RoboTHOR ReAct+Base+None 0.0 (full suite, round 1). The one-shot arms: DeliveryBench 12.44 (f01); ALFRED 0.50 (f01), 0.20 (f02, whose executor returned an empty response on 207 of 222 calls), and 0.30 (f03); MiniGrid 0.70 (f01, v2 draw), 0.80 (f02), and 0.80 (f03); RoboTHOR 0.00 (f01, completed by a byte-identical recovery run), 0.10 (f02), and 0.00 (f03). All evolution results are from the f02 campaigns. DeliveryBench candidate scores per round: 16.31/12.00/21.77, 15.25/7.17/13.86, 18.26/32.07/17.20, 21.87/15.00/20.86, 19.20/14.09/4.05 (labels a/b/c), read from the stored candidate histories.

A.5 LOG-ONLY ANALYSES: GATE DIAGNOSTICS, CODE, AND DESIGNER EFFORT

Source: analysis_results.json, produced by `figures/scripts/analyze_logs.py` *from the published snapshot without any new execution.* **Gate diagnostics.** The gate subset is the first four tasks in suite order: on MiniGrid the empty-room, fetch, lava-crossing, and multi-room tasks; on ALFRED simple placement, two-object placement, heat-then-place, and a movable receptacle; on RoboTHOR the apple, television, laptop, and alarm-clock episodes. The winners' gate SR against their SR

Table 8: Round winners of the four evolution arms (full-suite score; evolved methods in typewriter font; executor tokens of the winner's full evaluation).

Round	DeliveryBench (profit)	ALFRED (SR)	MiniGrid (SR)	RoboTHOR (SR)
R1	CoT+ActionLedgerV1: 21.8	ReAct+ Base+GroundedActionValidV1: 0.3	ReAct+SituatedDigestV1: 0.4	ReAct+Base (library): 0.0
R2	CoT+ActionLedgerV1+ ActionCheckV1: 15.2	same as R1: 0.3	ReActTraceV1+ SituatedDigestV1+ ActionGuardV1: 0.4	GoalReActV1+Base: 0.0
R3	ActionSelectCoT+ ActionLedgerV1: **32.1**	ReAct+ Base+ActionCommitV1: 0.3	...+CommitGuardV2: 0.3	CommitReActV1+Base: **0.1**
R4	same as R3: 21.9	ReAct+OutcomeTraceMemV1+ ActionCommitV1: **0.5**	...+CommitGuardV2: 0.4	GroundedReActV1+Base: 0.0
R5	ActionSelectCoT+ TrendLedgerV1: 19.2	ReAct+OutcomeTraceMemV2+ ActionCommitV1: 0.5	...+CommitGuardV3: 0.4	BudgetedReActV1+ DeadlineGateV1: 0.0
Exec. tok./eval	1.0–6.3M	4.5–5.5M	1.6–2.6M	1.7–4.5M
Designer out/cache-read	0.23M / 24M	0.37M / 41M	0.29M / 25M	0.35M / 33M

on the remaining six tasks was 0.75/0.17, 0.50/0.33, 0.50/0.17, 0.75/0.17, 0.50/0.33 on MiniGrid (rounds 1–5), 0.25/0.33, 0.00/0.50, 0.00/0.50, 0.25/0.67, 0.25/0.67 on ALFRED, and 0.00/0.00 in every RoboTHOR round except round 3 (0.25/0.00). All three candidates tied on the gate in 3 of 5 MiniGrid rounds, 1 of 5 ALFRED rounds, and 5 of 5 RoboTHOR rounds, so in those rounds the winner was chosen by proposal order; since the gate and the remaining tasks differ in difficulty, the score differences above do not by themselves show mis-ranking, and the direct evidence is that the gate provided limited discrimination between candidates in these rounds. **Task coverage.** MiniGrid: the union of tasks solved by the eight library configurations and by the five evolution rounds is the same six tasks; the one-shot draws solve all ten, and four tasks (four-rooms, goto-door, lava-crossing, and the second simple-crossing) are solved only by them. ALFRED: evolution rounds solve five tasks, one-shot draws seven; one task (a cool-then-place trial) is solved only by evolution and three (a movable-receptacle, a clean-then-place, and a heat-then-place trial) only by one-shot draws; a two-object and a second movable-receptacle trial are solved by no workflow. RoboTHOR: eight of ten episodes are solved by no workflow. DeliveryGym: of 32 validation shifts, 17 yield at least one delivery under every harness, 8 under none, and only 7 separate harnesses. **Evolved code.** The f02 campaigns produced 7, 7, 10, 11, and 8 evolved methods (DeliveryBench, ALFRED, MiniGrid, RoboTHOR, DeliveryGym) of 240–430 non-comment lines each on average, most inheriting from a library class or from an earlier evolved version; the one-shot agents are single programs of 1,780–2,360 lines with six to eight source files, six to eight test files, and at most two direct executor call sites. **Designer effort.** The evolution sessions contain 358–688 assistant turns with 160–270 shell commands, 16–22 file creations, and 19–32 edits over 1.5–5 days including evaluation waits; the one-shot sessions contain 111–296 turns with 60–124 shell commands over about five hours. **Compute on failures.** The best evolved harnesses spent 72–90% of executor tokens on episodes that ended in failure (ALFRED 76%, MiniGrid 82%, RoboTHOR 96%), the one-shot draws 42–55% on ALFRED and MiniGrid.

A.6 Design Cost

Figure 5 reports score and deployment cost (Section 6.4); Table 9 gives the per-arm accounting and Figure 8 adds the design and evaluation cost rows. Figure 7 places every evaluated MiniGrid and ALFRED harness on the accuracy–deployment-cost plane: the one-shot programs occupy the cheap corner, and on MiniGrid also the accurate one, while every harness that keeps the executor in the loop at each step sits in the library's cost band.

A.7 Pilot Campaign with Behavioural Digests and a Held-Out Suite

Before the final study we ran a pilot loop with a different feedback and selection design: one candidate per round, behaviour-only digests (failure modes, action histograms, quoted reasoning; no task solutions), a best-so-far gate with a regression tolerance and a task-flip noise floor, and a held-out suite scored for accepted rounds but never shown to the designer. Executor Qwen3.5-9B;

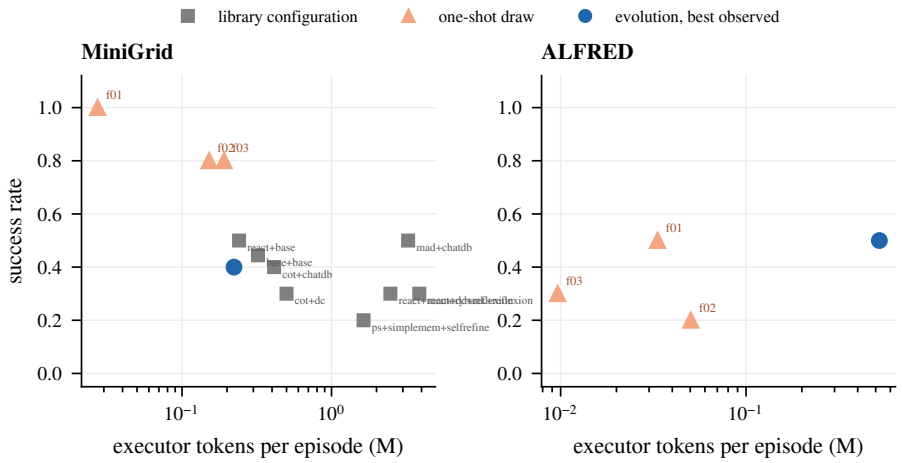


Figure 7: **Accuracy against deployment cost on MiniGrid and ALFRED.** Every evaluated harness under the shared protocol: library configurations (grey squares, MiniGrid calibration set), one-shot draws (triangles), and the best evolved harness (circle). On both suites the one-shot agents that moved perception, geometry, or planning into code are cheaper per episode than any harness that keeps the executor in the loop at every step, and on MiniGrid they are also more accurate.

Table 9: **Design, search, and deployment cost per evolution arm** (accounting snapshot). Search episodes count gate and complete-suite evaluations of all 15 candidates; evaluation tokens and hours cover every evaluation the campaign ran; deployment cost is the best observed harness's executor tokens per episode. A one-shot draw pays one design session (0.13–0.21M output tokens) and one evaluation (0.05–1.4 h).

Arm	Designer output / cache-read tokens	Search episodes	Executor tokens, all evaluations	Eval. hours (+queue)	Deploy tokens / episode
DeliveryBench	0.23M / 24M	30 (15 full×2)	37.6M	9.0 (+8.9)	1.57M
ALFRED	0.37M / 41M	110 (15×4 + 5×10)	70.4M	19.4 (+2.9)	0.52M
MiniGrid	0.29M / 25M	110	23.4M	7.9 (+8.1)	0.22M
RoboTHOR	0.35M / 33M	110	37.5M	20.8 (+3.2)	0.44M

search suite `babyai_goto` (36 BabyAI *GoTo* tasks); held-out suite `minigrid_navigation` (23 tasks). The search score rose from 38.9% to 52.8% (an authored 342-line ReAct variant that states the observation surface and reports the outcome of the last action) and 58.3% (memory switched off); a third candidate (an anti-oscillation action override) scored 50.0% and was reverted; the held-out score fell from 34.8% to 26.1% while the search score rose (Figure 9). The pilot motivated two elements that the final study kept, immutable candidate identity and evaluator-owned scoring, and a third that the final study applies after the campaign rather than during selection: the held-out evaluation of the frozen harnesses (Table 2), whose value this pilot demonstrates.

A.8 Repeated Evaluations of the Frozen Harnesses

Source tag: repeated executions of frozen programs after the study; no design, gate, or feedback step was involved. Each harness of Table 2 was re-executed with the study's own runner (`runner.unified`) on the study's suites, seeds, step caps, and scoring, against a separate vLLM deployment of the same Qwen3.5-9B checkpoint; the executions were operator-authorized and appear in the execution ledger. The frozen programs are the MiniGrid round-1 winner (ReAct with `SituatedDigestV1`: digest 2000 characters, two recent steps, trace length 8, untried actions listed), the DeliveryBench round-3 winner (`ActionSelectCoT` with three samples and action-name voting over `ActionLedgerV1` with a 12-step window), the library configurations ReAct+Base (MiniGrid) and CoT+MemoryBank (DeliveryBench), and the one-shot agents f02 and f03 (MiniGrid) and f01 (DeliveryBench), loaded from their finalized sources. Evaluations were sharded over episode workers as in the campaigns (five shards on MiniGrid, two on DeliveryBench). The executor temperature was 1.0 on MiniGrid and 0.1 on DeliveryBench for all three DeliveryBench harnesses, the value the DeliveryBench campaign used,

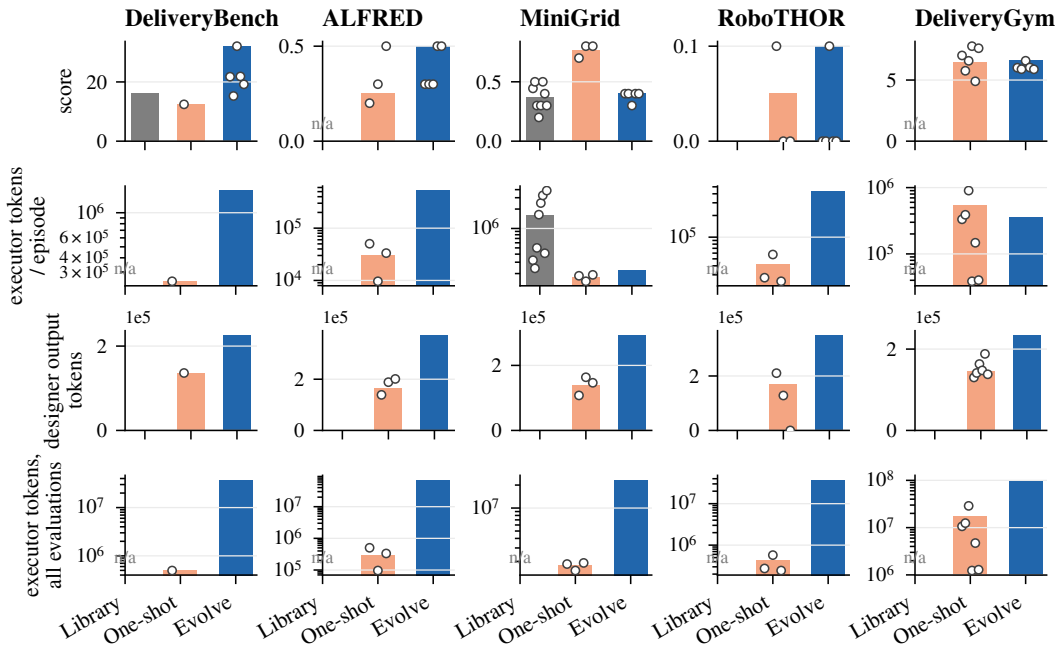
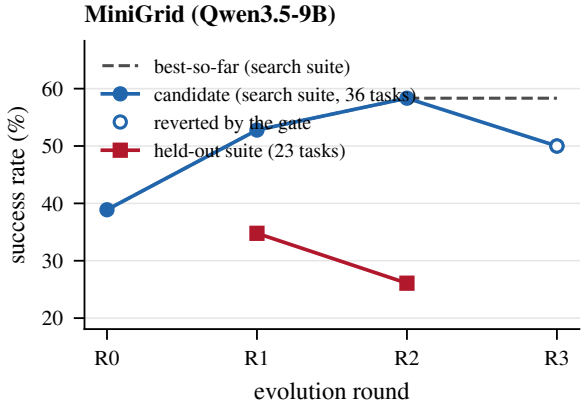


Figure 8: Score and the three cost terms per construction workflow: complete-suite score; executor tokens per episode of the final harness (deployment cost); designer output tokens (design cost); executor tokens over every evaluation the workflow ran (evaluation cost). The library's design and evaluation costs were incurred before this study and are not measured here, so its design and evaluation bars are omitted.



Round	Change (levers)	Search	Held-out
R0	ReAct + simple memory (baseline)	38.9	–
R1	authored `grounded_react` (instruction, control)	52.8	34.8
R2	memory simple → none (config.)	58.3	26.1
R3	anti-oscillation override (control)	50.0†	–

†reverted by the gate.

Figure 9: Pilot campaign: candidate and best-so-far search scores, and the held-out score of accepted rounds.

so the DeliveryBench repeats are a temperature-matched comparison. The held-out MiniGrid suite (`minigrid_heldout10`) keeps the task families and difficulty band of the development suite with disjoint environment–seed pairs: Empty-8x8 (seed 7), GoToObject-8x8-N2 (42), GoToDoor-6x6 (42), Fetch-6x6-N2 (42), SimpleCrossingS9N1 (7), LavaCrossingS9N1 (42), MultiRoom-N2-S4 (123), LavaGapS6 (42), DoorKey-6x6 (42), and SimpleCrossingS9N2 (7), each with a 100-step cap; none of these pairs appeared in any prompt, gate, or feedback. Per-episode outcomes, executor calls, and tokens of every repeat are released with the artifacts.

A.9 ONE-SHOT DRAWS, LIBRARY CALIBRATION, AND DELIVERYGYM RECORDS

Source tag: final controlled evaluation (one-shot arms and DeliveryGym), calibration set (library). One-shot draws (score; executor tokens per episode; wall time per episode): ALFRED f01 0.50 (0.03M, 310 s), f02 0.20‡ (0.05M, 319 s), f03 0.30 (0.01M, 27 s); MiniGrid f01 0.70 (0.18M, 933 s; v2 draw under the f02 protocol with a recorded pre-evaluation amendment), f02 0.80 (0.15M, 789 s), f03 0.80 (0.19M, 1082 s); RoboTHOR f01 0.00 (recovery), f02 0.10 (0.03M), f03 0.00 (0.06M); DeliveryBench f01 12.44 (0.25M). Draws f02 and f03 were generated from the same repository commit and prompt; f01 from an earlier one. MiniGrid library calibration on the identical suite (SR; tokens per episode): Base+Base 0.44 (9 episodes; 0.32M), ReAct+Base 0.50 (0.24M), CoT+DC 0.30 (0.50M), P&S+SimpleMem+SelfRefine 0.20 (1.64M), ReAct+MemoryBank+Reflexion 0.30 (2.48M), MAD+ChatDB 0.50 (3.25M), CoT+ChatDB 0.40 (0.41M), ReAct+DC+Reflexion 0.30 (3.87M). DeliveryGym (suite `deliverygym_val32`, 32 validation shifts, 60 turns, mean earnings): one-shot v1 draws 7.77 / 4.90 / 7.60 (v1 prompt), v2 draws 7.01 / 5.76 / 6.57; evolution rounds 6.01, 5.89, 6.58, 6.02, 5.88 (mean 6.08; rounds 1, 3, and 4 are ReActTrace+Base under different configuration identifiers and round 2 is ReActTraceV2, so the round-3 value is a higher observed score within the same method family rather than a new mechanism), over 15 gate and 5 complete-suite evaluations (280 episodes, 6.8 evaluation hours; 0.23M designer output tokens); rounds 1–4 ran the designer through the OpenRouter route and round 5 continued the same run in a Claude Code account session, with the four round summaries and the three most recent feedback packages supplied verbatim (recorded migration amendment); library ReAct+Base scored 8.33 on the four-shift gate only. Designer cost per run and evaluation hours are in the accounting tables released with the artifacts.

A.10 DELIVERYGYM PROTOCOL AND EXTERNAL REFERENCES

Source tag: confirmatory protocol; the executed runs are recorded in Appendix A.9. Facts taken from the benchmark paper (Kang et al., 2026a): 13 maps; 130 test shifts (100 in familiar cities, 30 in unseen cities); a 60-turn cap per shift; Waypoint and Any-Point movement modes; net income as the headline metric; three model repeats per shift in the official protocol; a privileged oracle that searches the simulator. Its main model table does not include Qwen3.5-9B, so published rows are external references (tier A) and cannot be compared with this paper's controlled rows (tier B) unless the environment commit, map assets, shift identifiers, movement mode, executor checkpoint, tool schema, observation fields, context and output limits, temperature, turn accounting, stop rule, error handling, and score aggregation all match. The tier-B rows of the confirmatory protocol are the official DeliveryAgent harness re-run under this executor, the initial AGENTSPEC library harness, FREEFORM-ONESHOT, AGENTSPEC-EVOLVE frozen before test, and FREEFORM-EVOLVE at matched budget. The designer may read only development cities, shifts, and logs; unseen cities must not appear in prompts, code, or readable task instances; the program and any cross-episode memory are frozen products; every primitive environment action counts against the same turn budget whatever the harness batches; any extra vision or planning model is part of the harness and its cost. Diagnostics to freeze before comparison: completed orders, timeout rate, resource exhaustion, repeated invalid actions, replanning count. Two extensions are kept separate: DeliveryBench→DeliveryGym zero-shot reuse of the evolved reasoning, memory, and reflection components through a pre-declared interface mapping (cross-version reuse, not cross-domain generality), and Waypoint→Any-Point, distinguishing re-search, frozen transfer, and independent evaluation.

A.11 HUMAN INTERVENTION LOG

Round budget set to 5 for all arms (applied uniformly while the arms were in progress; no completed round was re-run and no arm had reached round 6); ALFRED and RoboTHOR render GPU migrated from round 4, after which the renderer shared a GPU with the executor endpoint; the one-shot RoboTHOR f01 draw was completed by a byte-identical recovery run after an authorization-file race had left five tasks unrun; the one-shot MiniGrid f01 draw was taken under the f02 protocol (v2); the DeliveryGym designer route migrated for round 5. No intervention touched a candidate.

B CASE STUDIES: A MECHANISM-LEVEL AUDIT

The main text makes three claims about the generated harnesses: that a change the designer made can be located in one component and read against its execution record (Section 6.1), that the DeliveryBench advantage is consistent with the evolved harness's ledger and adjudication design (Section 6.3), and that the MiniGrid gap lies in what the executor could see (Section 6.2). This appendix audits each claim on the raw artifacts: the feedback the designer received, the code it wrote, and the executions that followed. Every excerpt below is verbatim from the released records or from the repeated executions of Appendix A.8.

Table 10: **Corpus behind the case studies.** Complete-suite evaluations per workflow in the three case environments, and executor calls per episode of the best evolved harness (range over the five round winners in parentheses) and of the one-shot programs.

Environment	Library evals	Evolution rounds	One-shot draws	Episodes / eval	Calls / ep., evolved	Calls / ep., one-shot
DeliveryBench	1	5 (15 candidates)	1	2	174 (52–326)	31
ALFRED	gate only	5 (15 candidates)	3	10	90 (77–93)	6–14
MiniGrid	8	5 (15 candidates)	3	10	63 (51–77)	22–24

B.1 ALFRED: INTERFACE COMPATIBILITY IS NOT STATE SEMANTICS

The ALFRED campaign is the clearest instance of a defect that a typed interface cannot catch and that the execution record can. Through round 3 the harness was the library's ReAct reasoner with the library's recency-window memory and an evolved reflection; the memory satisfied its contract (it stored each step's experience and returned the last five), yet the complete-suite score stayed at 0.30 for three rounds.

What the designer saw after round 3 (formal feedback, complete suite)

Harness: ReAct + Base memory + `ActionCommitV1`. SR 0.30, goal-condition rate 0.42, 808 executor calls over the ten episodes. Failure categories: **3 success, 3 terminated without reward, 4 truncated at the step limit**. The feedback also carried two bounded trajectories and, on disk, the library source; from these the designer established that ReAct renders no history of its own, so all trajectory context comes from the memory module, whose recency window restated the previous step's outcome one line late.

The defect, in the designer's own words (module docstring of `OutcomeTraceMemV1`, round 4, verbatim)

"The consequence is never recorded. The framework hands the memory channel a full transition, outcome flag and all, and the tail keeps the state the action was chosen in and drops what the action did. [. . .] What looks like the consequence belongs to the previous step. Many environments restate the last transition's outcome inside the next observation. Carried into a trace verbatim, that sentence lands on the line of the following action, so a step that failed reads as succeeded and a step that succeeded inherits the previous failure. A misaligned trace is worse than a silent one: the policy draws confident conclusions from it. [. . .] It re-proposes an action the environment refused eight steps ago, because nothing in what it can see says the refusal happened, and it abandons an approach that was working, because the failure text it can see is attached to the wrong line."

The repair (same docstring, verbatim), and how the lifecycle located it

"This module keeps the recency window exactly as it is – same fields, same order, same rationale, same observation – and makes the trace causally honest: each step's line ends with the outcome of that step, read from the transition the framework already provides [. . .]; a step that moved one of the environment's own progress counters says so, with the delta, so progress is attributed to the action that produced it; the previous transition's outcome sentence is removed from the stored observation, because the trace now carries that fact on the line it belongs to and carrying it twice is what makes it wrong." The change is confined to the memory component: the transition the lifecycle already hands to memory (Table 4) carried the outcome flag the new line needed, and the reasoner, perception, and loop were untouched. The module makes no executor calls of its own. The candidate passed 45 offline unit tests and the integrity and generality scans before evaluation.

What the record showed (round 4, complete suite)

Harness: ReAct + `OutcomeTraceMemV1` + `ActionCommitV1`. SR **0.50** (from 0.30), goal-condition rate 0.60 (from 0.42); failure categories **5 success, 1 terminated without reward, 4 truncated**. Terminations fell from three to one while truncations stayed at four: the aligned trace stopped the premature `Done` and the repeated refused actions, and left the search failures untouched. Round 5 kept the memory (a second version) and scored 0.50 again. All three round-4 candidates scored 0.00 on the four-task gate, so this winner was selected by proposal order; its complete-suite result is what the campaign recorded (Table 11).

Takeaway

The library memory was structurally valid and semantically wrong by one step. The typed lifecycle made the defect a property of one component, the execution record made its consequence visible as a change in failure categories, and the same record shows what the repair did not fix. This is the sense in which AGENTSPEC makes state semantics inspectable: it does not make components correct by construction, it makes their behaviour attributable.

B.2 DELIVERYBENCH: A COMPRESSED LEDGER AND ADJUDICATION ON DISAGREEMENT

The DeliveryBench winner (round 3) replaced the library's observation dumps with `ActionLedgerV1`, a one-line-per-step record of the executor's own decisions with repetition counts, and replaced plain chain-of-thought with `ActionSelectCoT`, which samples three chains and asks the executor to adjudicate only when they disagree. The excerpt below is the ledger exactly as the executor received it at step 30 of a repeated execution (Appendix A.8); the memory itself truncates each intent.

ActionLedgerV1 as rendered for the executor (repeated execution 1, episode 1, step 30; verbatim)

```
Action ledger: your own decision trace for the current episode, oldest first.
Lines are 'step | action you proposed | notes'.
(17 earlier steps omitted; 29 steps taken so far.)
  18 | PICKUP(orders=[1]) | intent: Step 1: I'm at Restaurant 3 (-424.45m, 53.15m) which is the pickup
      location for Order #1. The order status shows "Rea...
  19 | PLACE_FOOD_IN_BAG(bag_cmd="order 1: 1,2 -> A; 3 -> B") | intent: Step 1: I have Order #1 active with
      status "Picked up, waiting for delivery" and 3 items (PokeBowl, Salad, Cake) that...
  20 | MOVE(219.40m, -108.99m) | intent: Step 1: I have Order #1 active with status "Picked up, waiting for
      delivery". I'm currently at Restaurant 3 (-424.45m...
  21 | DROP_OFF(oid=1, method="leave_at_door") | intent: Step 1: I'm at the drop-off location (219.40m,
      -108.99m) for Order #1, which is ready for delivery. The order status...
  22 | VIEW_ORDERS() | intent: Step 1: I have no active orders and I'm currently at (219.40m, -108.99m) with
      65% energy and 67% battery. I need to f...
  23 | ACCEPT_ORDER(order_id=11) | intent: Step 1: I'm at (219.40m, -108.99m) with 65% energy and 67% battery,
       no active orders. I need to accept profitable ord...
  24 | MOVE(-288.53m, -16.58m) | intent: Step 1: I have an active order #11 with pickup at Restaurant 1
      (-288.53m, -16.58m) and dropoff at (-474.37m, 224.04m)...
  25 | WAIT(minutes=4) | intent: Step 1: I'm at Restaurant 1 (-288.53m, -16.58m) which is the pickup location
       for Order #11. The food is still being p...
  26 | PICKUP(orders=[11]) | intent: Step 1: I'm at Restaurant 1 (-288.53m, -16.58m) which is the pickup
      location for Order #11. The order status shows "R...
  27 | PLACE_FOOD_IN_BAG(bag_cmd="order 11: 1,2 -> A; 3,4 -> B") | intent: Step 1: I have Order #11 active
      with status "Picked up, waiting for delivery" and 1 minute remaining. I'm at Restaura...
  28 | MOVE(-474.37m, 224.04m) | intent: Step 1: I have Order #11 active with status "Picked up, waiting for
      delivery" and only 1 minute remaining. I'm curren...
  29 | DROP_OFF(oid=11, method="leave_at_door") | intent: Step 1: I'm at the drop-off location for Order #11
      at (-474.37m, 224.04m). The order status shows "Picked up, waiting...
Proposed repeatedly this episode: VIEW_ORDERS() x4.
```

The decision procedure, in the designer's own words (module docstring of `ActionSelectCoT`, round 3, verbatim)

"`ActionSelectCoT` *keeps the sampling and replaces the aggregator. When the samples disagree, it puts the distinct proposals side by side – each with the reasoning that produced it – and asks the model once which one to take. That is a strictly easier question than the original one: choosing among a few concrete, already-justified options requires no generation and no action-format compliance, only a comparison, which is the kind of judgement a small backbone is relatively good at. The adjudication is free whenever it is unnecessary. If every sample agreed there is nothing to choose between, so no call is made; the cost is paid only on genuine disagreement, which is exactly the signal that the decision was uncertain.*" This is why the evolved harness consults the executor 174 times per one-hour episode against 31 for the one-shot dispatch

program, whose fare, rating, and clock rules are encoded in code and which asks the executor only to choose among orders.

Takeaway

The DeliveryBench advantage rests on two mechanisms the designer wrote inside the typed space: a history the executor can read at one line per step, and a decision procedure that spends extra executor calls exactly where sampled decisions disagree. In the repeated executions the frozen harness earned 14.3, 19.1, and 20.0 per hour against 17.5, 13.0, and 10.9 for the one-shot program and 10.4 and 12.6 for CoT+MemoryBank, so the ordering of the campaign holds on average when the harness is re-executed.

B.3 MiniGrid: What the Executor Could See

On MiniGrid the evolved harnesses changed what the executor was told (a situated digest of visited situations and untried actions, commitment and action guards) but not what it could see. The library perception adapter passes the rendered partial view plus a few structured flags; the text the executor receives at a step of a repeated execution is reproduced below, and it describes no cell in view other than the one directly ahead.

Observation text received by the library harness's executor (ReAct+Base, Empty-6x6, step 3 of a repeated execution; verbatim)

```
Mission: get to the green goal square. Facing: right. Symbolic=AgentPos=(np.int64(3), np.int64(1)), Dir=right
       | Carrying=None | CurCell=empty, OnGoal=False | OnGoal=False | Front=empty, color=None
## Shared Episode Context
- Recent actions in this episode: move_forward -> move_forward. Mission: get to the green goal square
ObsSummary: Symbolic=AgentPos=(np.int64(3), np.int64(1)), Dir=right | Carrying=None | CurCell=empty, OnGoal=
      False | OnGoal=False | Front=empty, color=None. Facing right
```

What the one-shot program f02 does with the same rendered frame (its `navcore` package)

`tile_vision.decode_egocentric_view` slices the rendered frame into its tile lattice and describes each tile by dominant hue, coverage, and placement; its docstring states the design rule: *"this module holds no table of environment objects and no notion of what any colour means. It answers 'what does this patch of pixels look like', never 'what is it'. Attaching meaning to a descriptor is done elsewhere, by learning the association between a descriptor and the label the environment itself reports for that cell when the agent inspects it."* `spatial_memory.SpatialMemory` keeps a persistent map of observed and labelled cells across steps; `dynamics.ActionModel` records the effect of each action and `IneffectiveMemory` the actions known to be useless in a situation; `explore` plans on the map; and one `LLMInterface` asks the executor for instruction interpretation and target guesses, 22–24 times per episode. The program is 2,312 lines across eight source files, and every executor call goes through its single `LLMInterface` wrapper.

Takeaway

Every one-shot program on MiniGrid inverted the renderer into cells and planned on a map; none of the evolvable components in this study (reasoning, memory, reflection) could change what perception emitted. The gap between 0.40 and 0.70–0.80, reproduced under repeated and held-out execution (Table 2), is therefore a statement about the editing scope of this study, and the perception adapter is the component the substrate would expose next.

B.4 The ALFRED Lineage End to End

Failure: the round-1 winner (library ReAct and memory plus an evolved reflection that validates a proposed action against the visible scene) succeeded on three simple pick-and-place tasks only; the policy issued `Done` prematurely (one episode forfeited 84 of 111 steps at zero goal completion) although the prompt forbade it, and four of ten episodes exhausted the reflection module's repair cap. *Evidence the designer saw:* per-task outcomes, the failure categories, two trajectories, and the library code, from which it established that ReAct renders no history of its own, so all trajectory context comes from the memory module, whose recency window restated the previous step's outcome one line late. *Diff:* round 4 added `OutcomeTraceMemV1` (keeps the window byte-for-byte, appends each step's own outcome read through generic interfaces, removes the late restatement); two variants

added a ledger of repeatedly failed actions or truncated observations. *Checks:* 45 offline unit tests, integrity and generality scans. *Gate and complete suite:* all three candidates scored 0.00 on the gate, the winner was chosen by proposal order, and its complete-suite score rose from 0.30 to 0.50 (goal completion 0.42 to 0.60; terminations without reward 3 to 1; truncations unchanged at 4). *Adoption:* the round-5 successor kept the memory and scored 0.50 again. *Estimator:* development-suite scores; the frozen winner's repeated and held-out execution is not part of this lineage.

Table 11: **ALFRED lineage by round.** Gate SR lists the three candidates' scores on the four-task gate in proposal order; suite SR is the round result on the complete suite; categories are success / terminated without reward / truncated at the step limit over ten episodes.

Round	Winner	Methods added this round	Gate SR (a/b/c)	Suite SR	Categories
1	ReAct + Base + `GroundedActionValidV1`	`SpatialEpisodicMemV1`, `GroundedActionValidV1`	0.00 / 0.25 / 0.00	0.30	3 / 4 / 3
2	ReAct + Base + `GroundedActionValidV1`	`ProgressBudgetMemV1`, `GroundedActionValidV2`	0.00 / 0.00 / 0.25	0.30	3 / 3 / 4
3	ReAct + Base + `ActionCommitV1`	`ActionCommitV1`	0.00 / 0.25 / 0.00	0.30	3 / 3 / 4
4	ReAct + `OutcomeTraceMemV1` + `ActionCommitV1`	`OutcomeTraceMemV1`	0.00 / 0.00 / 0.00	**0.50**	5 / 1 / 4
5	ReAct + `OutcomeTraceMemV2` + `ActionCommitV1`	`OutcomeTraceMemV2`	0.25 / 0.00 / 0.00	0.50	5 / 1 / 4

C Controlled Component Studies

Source tag: component study. This appendix reports the controlled component study that preceded the evolution experiments. Every harness here is a configuration of the module library of Appendix E, and because backbones, task subsets, thinking modes, and step budgets differ from the protocol of Section 5.1, these numbers are kept apart from Table 1. The study establishes that module effects are environment-dependent and non-additive, which is what makes the search of Section 4 necessary, and it supplies the mechanistic context against which the evolved harnesses are read in Section 6.1. Abbreviations: R=ReAct, CDB=ChatDB, SMem=SimpleMem, MB=MemoryBank, DC=DynamicCheatsheet, OC=OpenClaw, P&S=Plan-and-Solve, SR=SelfRefine, Rfx=Reflexion.

C.1 Overall Performance Across Tasks

Experimental Setup. The component study's primary experiments are on DeliveryBench under the **1-hour setting**, a long-horizon and resource-constrained testbed for evaluating how agent modules affect embodied decision-making. We compare configurations along three axes—reasoning, memory, and reflection—across both open-source and closed-source backbones. For each backbone, we start from a lightweight base agent with simple memory and no additional reasoning or reflection, then swap in different module variants under a unified protocol. We also evaluate representative configurations on MiniGrid, ALFRED, and RoboTHOR to test whether the trends transfer across environments with different horizons, observation modalities, and task structures.

Scaffolding narrows the backbone gap. As shown in Figure 10(a), stronger backbones generally achieve higher best-case DeliveryBench performance, with Qwen-27B and closed-source models outperforming smaller Qwen variants. However, the gap is not determined by scale alone. Well-matched modular configurations allow smaller or open-source models to approach stronger models: Qwen-27B with ReAct+MemoryBank performs strongly, and Qwen-9B also benefits from memory- and reasoning-augmented variants. This suggests that external scaffolding can compensate for weaker native long-horizon reasoning, especially when it provides structured state tracking or reusable task guidance.

Memory helps when it is structured. Figure 10(a) shows that memory can substantially improve DeliveryBench performance, but its benefit depends on how information is organized and retrieved. Structured, action-oriented memories such as DynamicCheatsheet and MemoryBank provide compact guidance useful for the next decision, while less selective memory may introduce stale or weakly relevant context. Thus, effective memory is not simply longer context; it must preserve task-relevant

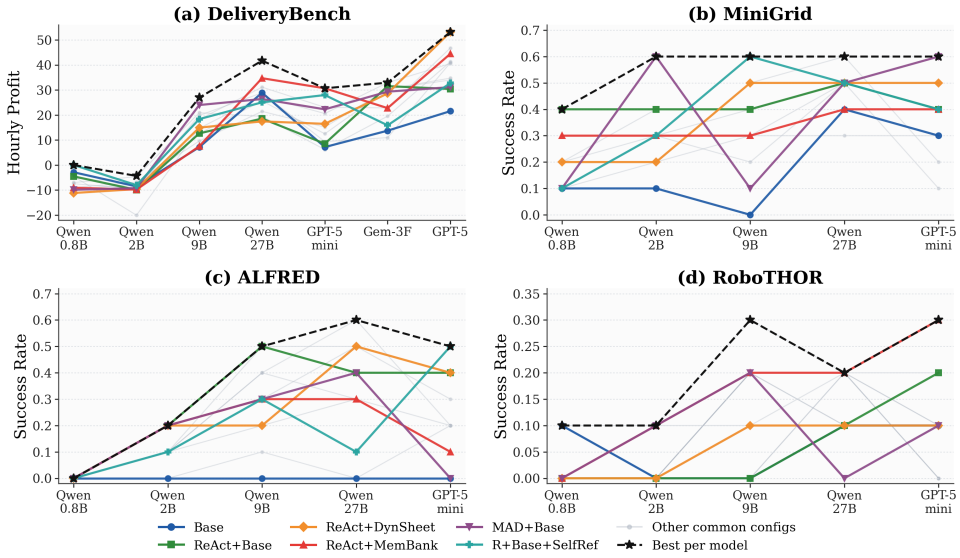


Figure 10: **Main results across four interactive-agent benchmarks.** Colored curves denote representative modular configurations, gray curves denote additional common configurations, and the dashed star-marked curve shows the best configuration for each backbone. DeliveryBench reports hourly profit; the other benchmarks report success rate. Higher is better. Full results are provided in the released artifacts.

experience in a concise and controllable form. Across the backbones of Figure 10(a), the same memory replacement moves reasoners by different amounts, which indicates that memory gains are mediated by the reasoner that consumes them.

The best module depends on the environment. The cross-benchmark results in Figure 10(b–d) show that module effectiveness varies across environments. In MiniGrid, where tasks are shorter and more symbolic, reasoning-oriented configurations often match the best envelope, while memory gains are less consistent. In ALFRED, both memory and reasoning matter because agents must maintain coherence over multi-step household instructions. In RoboTHOR, success remains lower and the best configuration varies across backbones, suggesting additional bottlenecks from perception, navigation, and long-horizon recovery. Overall, modules should be selected based on the dominant failure mode: reasoning improves local decision structure, memory supports long-horizon state tracking, and reflection helps when errors can be corrected through explicit revision.

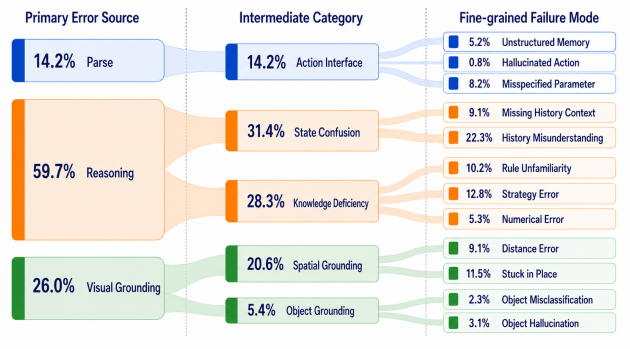


Figure 11: Failure taxonomy of the component study across the four environments (library harnesses; counts and denominators in the released artifacts). It links each environment's bottleneck to a writable component family and is the reference against which evolved changes are read.

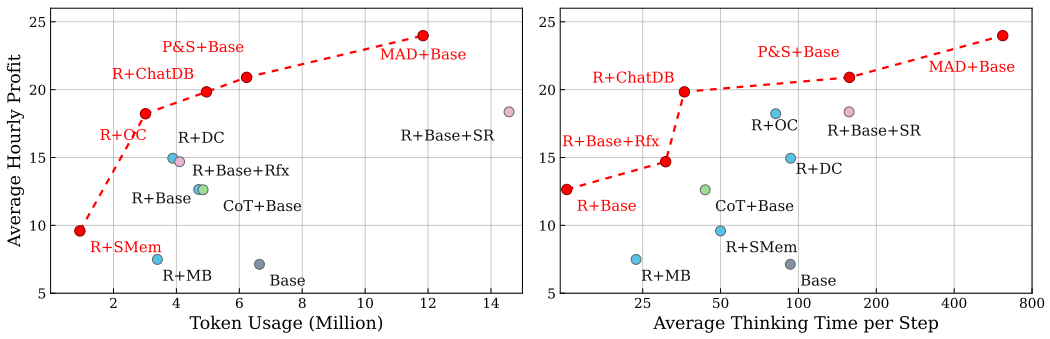


Figure 12: Pareto frontier of Qwen3.5-9B across different module combinations, showing the trade-off between (a) Profit vs. Token Usage and (b) Profit vs. Thinking Time (in log axis).

C.2 How Modules Interact

Experimental Setup. We further analyze module interactions on DeliveryBench. Since module effects in long-horizon embodied tasks are rarely independent, we compare reasoning, memory, and reflection combinations under a unified setting to identify helpful pairings and harmful mismatches.

Planning benefits from abstract memory. Across the evaluated configurations (Figure 10), ReAct and Plan-and-Solve gain little from low-level memories such as Base and ChatDB, which mainly return raw observations or trajectory fragments, and more from abstract memories such as SimpleMem, DynamicCheatsheet, and MemoryBank, suggesting that planning-oriented reasoning benefits more from distilled strategies than from exact historical states.

Multi-granularity memory transfers across reasoners. Among the evaluated configurations MemoryBank performs strongly across reasoning methods because it combines raw trajectories, experience summaries, and higher-level environmental insights. This lets each reasoning strategy use memory at the granularity it needs, making the module robust to different downstream reasoning styles.

Multi-agent reasoning tolerates weaker memory. Across the evaluated memory settings, multi-agent reasoning remains effective with low-level memory, showing lower sensitivity to memory quality than single-pass reasoning; error correction and complementary perspectives help maintain decision quality when retrieved memory is coarse.

Reflection is a correction layer with a cost. Across the evaluated pairs, reflection yields its largest gains for weak reasoning–memory pairs, smaller gains for strong ones, and losses in some cases, while every configuration adds calls per step; it re-evaluates candidate actions against state feedback and recent failures, reducing local execution errors before they accumulate.

C.3 Efficiency–Performance Trade-off

Motivation. Performance alone does not fully characterize embodied agents, since stronger module combinations may introduce substantial inference overhead over long-horizon interactions. We therefore evaluate each configuration by both effectiveness and efficiency, using *total token usage* and *average thinking time per step* as the main efficiency metrics.

More computation is not always better. As shown in Figure 12a, configurations with similar token budgets can achieve very different profits, while some higher-cost methods offer limited additional gains. Several ReAct-based variants lie on the Pareto frontier, suggesting that strong performance can often come from better module composition rather than simply larger inference budgets. Thus, token efficiency depends not only on reasoning strength, but also on whether reasoning and memory are well matched.

Latency depends on alignment, not just deliberation. A similar pattern appears in Figure 12b. Higher-profit configurations do not always require proportionally longer thinking time: some achieve strong returns with moderate latency, whereas others spend more time without comparable gains. This shows that useful computation is task- and module-aligned computation, not merely more deliberation.

C.4 Detailed Efficiency Analysis

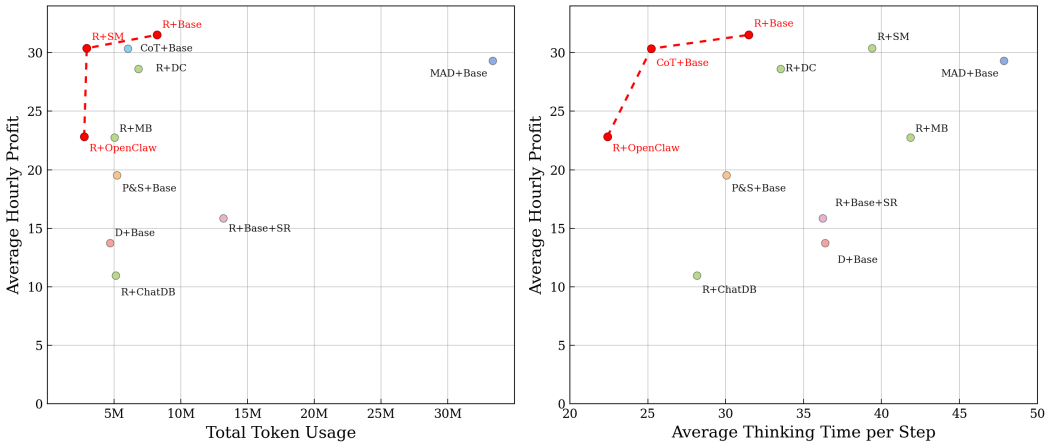


Figure 13: Gemini-3-Flash Pareto plots.

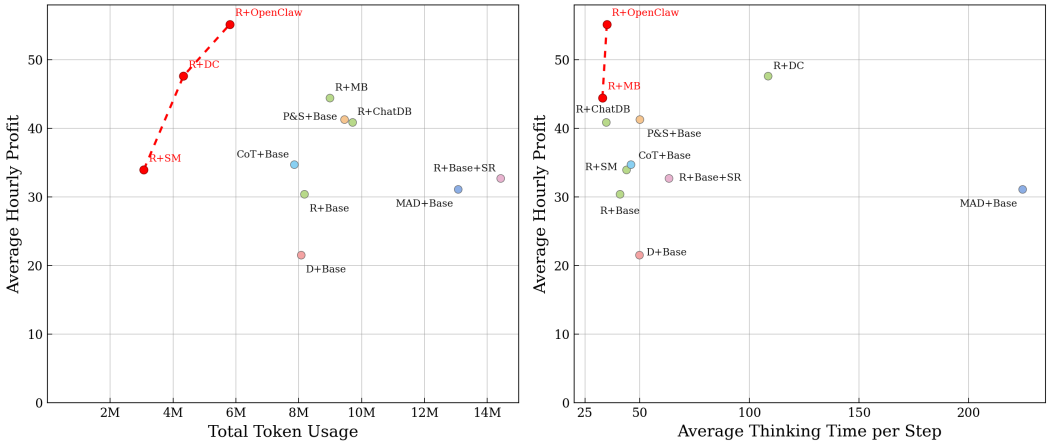


Figure 14: GPT-5 Pareto plots.

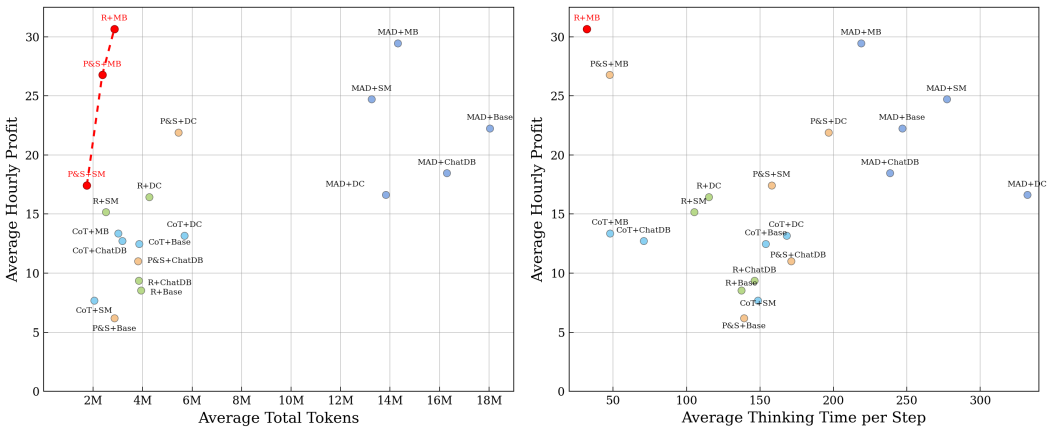


Figure 15: GPT-5 mini Pareto plots.

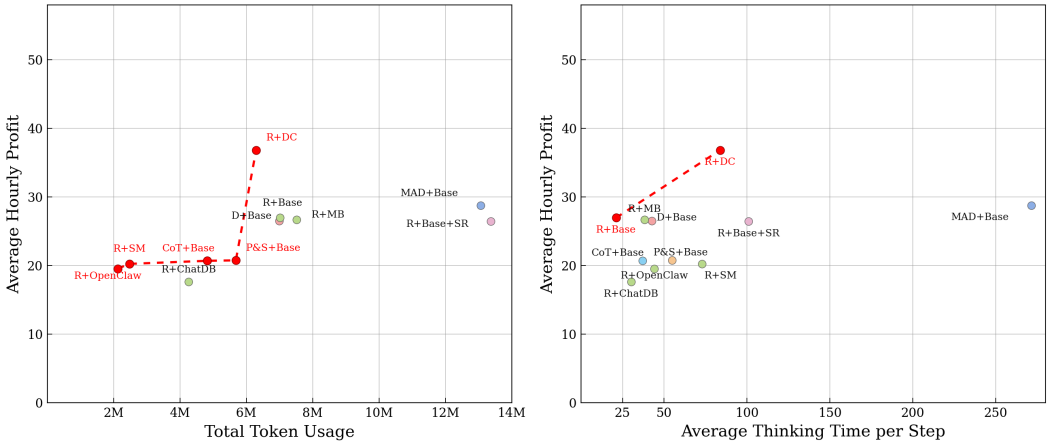


Figure 16: Qwen3.5-397B Pareto plots.

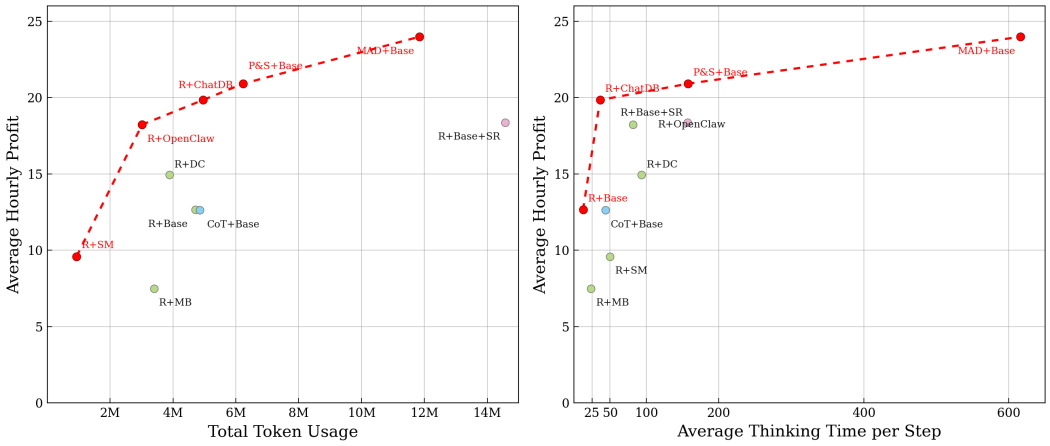


Figure 17: Qwen3.5-9B Pareto plots.

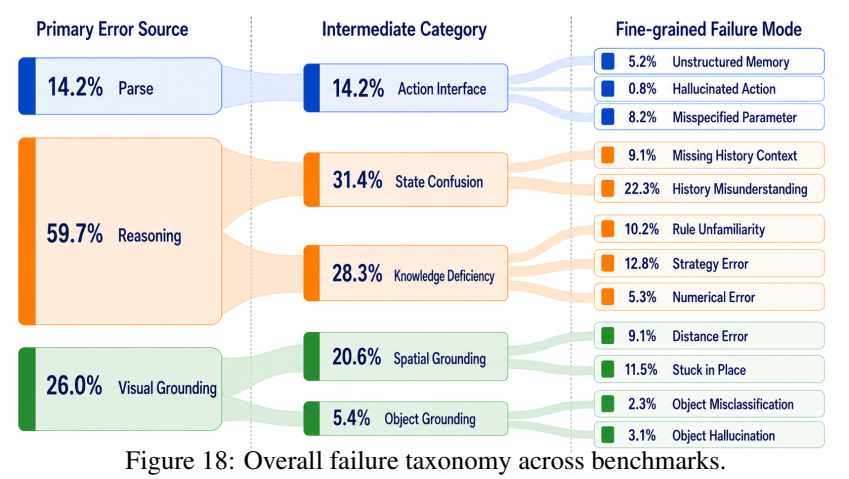


Figure 18: Overall failure taxonomy across benchmarks.

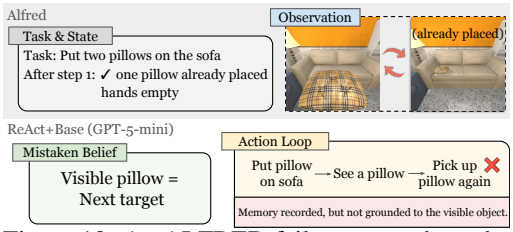


Figure 19: An ALFRED failure case where the agent fails to align recent history with the current observation, leading to a repeated-action loop.

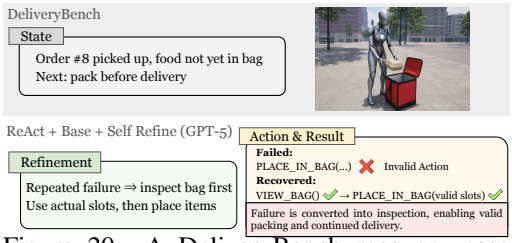


Figure 20: A DeliveryBench recovery case where Self-Refine turns repeated bagging failures into an inspect-and-repair process.

C.5 Case Study and Error Analysis

Many failures come from losing the task state. Figure 18 shows that failures across benchmarks often involve history misunderstanding: the agent fails to track the current state from past interactions, mistakes completed progress for the current objective, and falls into repetitive loops. Beyond this shared pattern, RoboTHOR and ALFRED failures concentrate more on visual grounding, while MiniGrid and DeliveryBench errors more often reflect poor adaptation to environment rules, action constraints, and long-horizon strategy, as detailed in the released artifacts.

Memory tracks progress; reflection repairs execution. Figure 19 illustrates a representative ALFRED failure caused by state confusion. Although one pillow has already been placed, the agent treats the visible pillow as the next target and repeatedly alternates between picking and placing actions, forming a loop. This suggests that lightweight episodic memory may not reliably distinguish completed progress from visually similar current observations. DeliveryBench exposes a different challenge: its action interface requires precise parameterization, and agents often fail by issuing invalid or over-compressed commands. Figure 20 shows that Self-Refine mitigates this by redirecting the agent to inspect the bag state, recover missing information, and resume with valid actions. Together, these cases show that memory is crucial for long-horizon progress tracking, while reflection helps diagnose invalid assumptions and repair failed action plans.

The two cases in Figures 19 and 20 come from hand-configured harnesses of this study (a lightweight episodic memory on ALFRED and Self-Refine reflection on DeliveryBench); they illustrate failure modes and runtime correction, and are not outputs of an evolution campaign.

D Extended Related Work

Table 12: **Nearest automated-design systems, by the properties that matter for harness evolution.** Entries summarize each paper's stated design; see the cited papers for details. What distinguishes AgentSpec is the embodied runtime contract (what each component may read, what state persists across steps and resets per episode, and how action outcomes return to memory and reflection), which is what lets a change to one component be located and read against its execution record.

System	What is edited	Declared component state / contract	Feedback to the designer	Designer runs the benchmark?	Reported estimator
AgentSquare	module selection and recombination in four families, plus new module code generated under standard I/O interfaces	module I/O signatures	task score	yes (search loop)	best found on the search tasks
ADAS	whole agent programs in code	none beyond a forward function	task score	yes (meta-agent loop)	search-set and held-out accuracy
AFlow	code-level workflow graphs	operator nodes	execution feedback and score	yes (MCTS loop)	best on validation, test reported
AgentSpec (this paper)	reasoning, memory, reflection implementations behind typed contracts; perception, action adapter, loop fixed	typed inputs/outputs, per-step and per-episode lifecycle, outcome return to memory and reflection, protected paths	bounded per-task feedback, failure categories, trajectories	no: evaluator-issued authorization	development-suite best-of-R, round trace, h_0, frozen-harness repeats, held-out suite (MiniGrid)

LLM Reasoning. Beyond the prompting and search-based reasoning methods cited in the main text, a wide range of additional reasoning strategies have been developed. Tool-augmented, logic-aided, and backward-chaining methods ground reasoning in external computation or formal inference (Kazemi et al., 2023), while constrained generation enforces structural validity during decoding (Banerjee et al., 2025). Self-correction is a particularly active area: iterative critique-and-revision (Welleck et al., 2023; Havrilla et al., 2024; Gao et al., 2024a), self-rewarding correction (Xiong et al., 2025b), cooperative reasoning across models (Yang et al., 2026a), and adversarial self-play (Li et al., 2026a) all enable models to refine their own outputs, complemented by process-level verification (Lightman et al., 2024). Multi-agent approaches leverage inter-agent debate and collaborative training (Du et al., 2024; Motwani et al., 2025; Yang et al., 2026b; 2025). Meta-level methods allow models to compose their own reasoning structures (Zhou et al., 2024b), treat reasoning as structure-aware planning with world models (Xiong et al., 2025a; Xiang et al., 2025), incorporate contrastive objectives into Monte Carlo tree search (Gao et al., 2024b), or adapt reasoning depth to task complexity via cognitive-inspired sketching or efficiency analysis (Sprague et al., 2025; Ning et al., 2024; Aytes et al., 2025). Latent reasoning methods bypass discrete token generation entirely, operating in continuous latent space (Hao et al., 2025) or via latent diffusion for iterative refinement and diverse solution exploration (Kang et al., 2026b), while GFlowNet-based training models reasoning as flow on a DAG to promote diverse generation (Yu et al., 2025; Kang et al., 2025). Reasoning has also been extended to embodied action selection (Li et al., 2024a), grounded simulation (Liu et al., 2023b), deep reasoning for translation (Wang et al., 2025b), and interleaved reasoning-memorization via self-notes (Lanchantin et al., 2023). In AgentSpec, all of these are interchangeable instantiations of a single reasoning interface, enabling controlled comparison within identical agent pipelines.

Memory for LLM Agents. Beyond the memory categories surveyed in the main text, database-backed approaches store dialogue history for SQL-style retrieval (Hu et al., 2023; Modarressi et al., 2023), and long-context memory extensions augment or replace the context window itself (Liu et al., 2024; Xiao et al., 2024; Yang et al., 2024a). Hierarchical and graph-based methods organize experience into multi-level semantic structures, temporal knowledge graphs, or event-centric logic maps (Li et al., 2024b; Jiang et al., 2026; Hu et al., 2026; Anokhin et al., 2025). Procedural memory systems store reusable action sequences, trajectory exemplars, or cross-domain skills for

transfer (Fang et al., 2026b; Han et al., 2025; Tang et al., 2025; Xiao et al., 2025; Forouzandeh et al., 2026; Zheng et al., 2024; Wang et al., 2025d; Liu et al., 2025b; Zhang et al., 2026a), while self-organizing and evolving designs adaptively restructure memory through cognitive-inspired consolidation, RL-based construction, or meta-evolution (Zhang et al., 2025b; Tian et al., 2026; Qian et al., 2025; Cao et al., 2026; Wei et al., 2025; Yan et al., 2026). Plug-and-play and parameter-efficient memory modules enable domain adaptation without retraining the base model (Cao et al., 2025; Bini et al., 2025; Packer et al., 2023). Episodic and social memory grounds agents in persistent interaction histories and simulated human behavior (Park et al., 2023; Li et al., 2025b; Zhong et al., 2024; Zhang et al., 2025c). Memory maintenance and semantic anchoring address staleness, coherence, and linguistic grounding over extended interactions (Chen et al., 2025; Chatterjee & Agarwal, 2025; Wang et al., 2026; 2025c). Multimodal and video-oriented memory extends these ideas to visual and lifelong learning streams (Yeo et al., 2026; Latimer et al., 2025; Liu et al., 2025a). A growing line of work further recognizes that memory and reasoning are mutually dependent: recall-and-post-thinking mechanisms augment reasoning with long-term memory (Liu et al., 2023a), reusable thought templates and evolving knowledge bases bridge past reasoning into current decisions (Yang et al., 2024b; Suzgun et al., 2026; Qian et al., 2025), hierarchical working memory and structured context manage cognitive load during multi-step reasoning (Hu et al., 2025a; Wang et al., 2025a), and reasoning-memory co-evolution allows agents to jointly improve both capabilities (Ouyang et al., 2026; Ho et al., 2025). In AGENTSPEC, these diverse memory designs are exposed through a shared module interface, enabling systematic evaluation of how storage, retrieval, and update choices—and their interaction with reasoning—shape agent performance.

Reflection and Self-Improvement. Reflection mechanisms range from step-level critique-and-revision (Paul et al., 2024) to trajectory-level self-critique that stores verbal reflections from failed episodes (Yao et al., 2024),reflection through trial-and-error driven test-time planning (Hong et al., 2026) and experience replay approaches that extract and internalize transferable insights from past trajectories (Zhao et al., 2024). Experience-driven lifelong learning enables agents to self-evolve through accumulated interaction (Wu et al., 2025; Chen et al., 2026). In AGENTSPEC, reflection is a dedicated module decoupled from reasoning and memory, enabling controlled ablation of its marginal contribution and interaction effects with other components.

E MODULE IMPLEMENTATIONS

All components listed in this appendix are human-written and form the initial library that LIBRARY-FIXED selection draws from and that the designer inherits at round 0. Components authored by the designer during evolution live in the round directories of Appendix A and are marked as designer-authored wherever they appear.

This section details the module implementations instantiated in AGENTSPEC. For each module family, we list the representative methods included in our study, summarize their operational characteristics, and provide the corresponding references. All implementations follow the common interfaces introduced in the main paper, enabling controlled comparisons across reasoning, memory, and reflection mechanisms.

E.1 REASONING METHODS

Table 13 reports the reasoning strategies included in our implementation suite. The selected methods span linear decomposition, explicit planning, search-based exploration, consistency-based aggregation, and multi-agent deliberation, covering the main design choices for structuring intermediate inference in LLM agents.

E.2 MEMORY METHODS

Table 14 summarizes the memory mechanisms implemented in AGENTSPEC. These methods cover short-context buffers, vector- and database-backed retrieval, hierarchical and graph-structured stores, procedural memory, and adaptive long-term memory systems, providing a broad basis for evaluating how storage, retrieval, and update policies affect agent behavior.

Table 13: Reasoning module implementations supported in AGENTSPEC.

Method	Description	Reference
ReAct	Interleaves reasoning and acting over multiple Thought–Action–Observation cycles, grounding decisions in environmental feedback.	(Yao et al., 2023b)
CoT	Generates explicit intermediate reasoning steps before selecting an action, decomposing complex decisions into a chain of thoughts.	(Wei et al., 2022)
Plan-and-Solve	Produces a high-level plan before execution, then follows the plan step by step to complete the task.	(Wang et al., 2023a)
Tree of Thoughts	Expands reasoning into a tree structure, exploring multiple thought branches and evaluating candidates via lookahead search.	(Yao et al., 2023a)
LATS	Combines language model reasoning with Monte Carlo Tree Search, using execution feedback to guide trajectory exploration.	(Zhou et al., 2024a)
RAP	Treats reasoning as planning via MCTS, building a world model to simulate and evaluate action sequences before committing.	(Hao et al., 2023)
Self-Consistency	Samples multiple independent reasoning paths and aggregates their outputs to select the most consistent answer.	(Wang et al., 2023b)
MAD	Introduces critique and disagreement among several language agents through structured debate rounds before reaching a final decision.	(Du et al., 2024)

Table 14: Memory module implementations supported in AGENTSPEC.

Method	Description	Reference
Base	Maintains a rolling buffer of recent transitions appended directly to the prompt context.	—
A-Mem	Maintains memory notes with semantic metadata, LLM-powered content analysis, and relationship management via ChromaDB hybrid search.	(Xu et al., 2025)
ACE	Agentic Context Engine applying iterative reflection, LLM-guided curation, and deduplication to maintain a compact, non-redundant memory store.	(Zhang et al., 2026b)
Buffer of Thought	Maintains a reusable buffer of high-level thought templates distilled from past reasoning traces, enabling retrieval-augmented thought reuse at inference time.	(Yang et al., 2024b)
CAM	Organizes experience into a hierarchical semantic structure using incremental overlapping clustering with LLM-guided pruning and coherent summarization.	(Li et al., 2025a)
ChatDB	Stores dialogue history in a structured database, enabling SQL-style retrieval of past interactions.	(Hu et al., 2023)
DC	Dynamic Cheatsheet that evolves a concise knowledge base via LLM curation and vector-store retrieval at test time.	(Suzgun et al., 2026)
GMemory	Three-tier hierarchical graph memory spanning an interaction graph for trajectory condensation, a query graph for task retrieval, and an insight graph for high-level insight management.	(Zhang et al., 2025a)

continued on next page

Table 14: Memory module implementations supported in AGENTSPEC (continued).

Method	Description	Reference
LangMem	LangChain-based memory infrastructure supporting episodic, semantic, and procedural memory types with structured storage and retrieval.	(LangChain AI, 2025)
MemGPT	Manages memory through a paged context window with explicit in-context and archival storage layers, enabling unbounded long-term memory.	(Packer et al., 2023)
LightMem	Manages memory through a multi-stage pipeline involving normalization, pre-compression, topic segmentation, LLM-based extraction, embedding, storage in Qdrant, and retrieval.	(Fang et al., 2026a)
Mem0	Adaptive personal memory with self-updating storage integrating short-term and long-term retrieval across user sessions.	(Chhikara et al., 2025)
MemoryBank	Long-term memory bank that stores and retrieves past experiences using similarity-based lookup to inform future decisions.	(Zhong et al., 2024)
MIRIX	Multi-agent memory system using event-time storage and context-aware retrieval via a dedicated Mirix client.	(Wang & Chen, 2025)
OpenClaw	Context management module that maintains procedural memory comprising learned policies and lessons, and augments agent prompts with relevant skill documents at each step.	(OpenClaw, 2026)
Generative Agent Memory	Scores memory entries by combining recency, semantic relevance, and reward utility into a unified ranking function.	(Park et al., 2023)
SimpleMem	Employs a three-stage pipeline: semantic compression, vector storage, and hybrid retrieval across semantic, keyword, and structured indexes.	(Liu et al., 2026)
Zep	Models memory as a temporal knowledge graph spanning episodes, entities, and higher-level communities.	(Rasmussen et al., 2025)

E.3 REFLECTION METHODS

Table 15 presents the reflection mechanisms included in our framework. These methods operationalize self-improvement at different temporal scales, from step-level critique and revision to episode-level retrospective analysis and reuse of lessons across subsequent trials.

Table 15: Reflection module implementations supported in AGENTSPEC.

Method	Description	Reference
Self-Refine	Performs iterative critique-and-revision at the step level: a critic evaluates the current output and proposes improvements until the answer stabilizes or no further improvement is detected.	(Madaan et al., 2023)
Reflexion	Stores verbal reflections from failed episodes and reuses them in subsequent trials, enabling trajectory-level self-correction across episodes.	(Shinn et al., 2023)
Retroformer	Retrospectively analyzes full trajectories after episode completion to distill lessons that improve later planning and decision-making.	(Yao et al., 2024)

F Detailed Evaluation Settings

This appendix has two parts. Subsections F.1.1–F.3.1 document the environment settings of the component study (Appendix C), including its own task subsets and repeat counts; Subsection F.4 lists the frozen suites of the workflow study, which are the only settings behind Tables 1 and 2.

F.1 DeliveryBench

F.1.1 DeliveryBench

DeliveryBench Mao et al. (2025) is a city-scale embodied benchmark in which the agent acts as an autonomous food courier in procedurally generated 3D cities. Unlike short-horizon embodied tasks, DeliveryBench emphasizes long-horizon decision making under realistic operational constraints. The agent must continuously choose profitable orders, travel between restaurants and customers, execute pickup and drop-off operations, and manage limited resources such as time, energy, and transportation cost. The benchmark therefore evaluates whether an agent can sustain constraint-aware planning over extended interactive trajectories rather than merely selecting locally plausible actions.

Task Setting. The original DeliveryBench benchmark is built around a profit-earning courier task in urban environments, with diverse road layouts, functional locations, transportation options, and realistic resource dynamics. At each step, the agent must reason over its current operational state, currently active orders, spatial context, and delivery constraints, then choose the next executable delivery action. In our framework, DeliveryBench is instantiated as a long-horizon single-agent environment under the same high-level objective of maximizing delivery profit. For the main study in this paper, we evaluate agent modules under the 1-hour setting, which provides a controlled testbed for comparing reasoning, memory, and reflection designs in a realistic resource-constrained scenario.

Evaluation Protocol. Within AGENTFACTORY, DeliveryBench is executed through the unified runner rather than through a benchmark-specific standalone pipeline. Episodes are evaluated over long-horizon interactions and terminate when benchmark lifecycle limits are reached or when the framework step budget is exhausted. This setup preserves the underlying DeliveryBench objective while allowing all agent variants to be compared under the same top-level execution interface. We use this shared framework to isolate the contribution of modular components without changing the surrounding evaluation pipeline.

Metrics. Following the benchmark goal, the main metric reported in the paper is *hourly profit*, which directly measures how effectively an agent converts long-horizon decisions into net delivery earnings. In the main DeliveryBench table, we report the mean hourly profit over three independent runs for each agent configuration. In addition to this headline metric, the framework-side evaluator also retains a normalized *score* derived from hourly profit and can record finer-grained delivery diagnostics when available, such as order-quality, time-efficiency, on-time, and resource-related indicators. In the main text, however, we use hourly profit as the primary metric because it best captures the benchmark's end objective and provides the most interpretable comparison across module combinations.

Unified Agent Input. To support cross-benchmark modularity, we expose DeliveryBench to the agent through a unified input interface rather than passing the original benchmark prompt format directly to each reasoning method. Concretely, the perception layer organizes the current observation into four conceptual components: *Image*, *Instruction*, *State Info*, and *Action Schema*. The *Instruction* component encodes the benchmark-level courier objective and task context. The *State Info* component repackages benchmark-native textual context into a structured agent-facing representation, including the current operational status, active orders, map-related context, recent actions, and recent execution feedback. When visual rendering is available, the *Image* component provides environment images through the same shared visual interface used by other embodied benchmarks. The *Action Schema* component normalizes benchmark actions into a standardized framework-facing action list with textual descriptions, so downstream reasoning modules interact with DeliveryBench through the same abstract interface used for other environments.

Input/Memory Interface. This unified representation is designed to stay compatible with the original DeliveryBench philosophy, which combines persistent task instructions with dynamic state and map context, while still making benchmark-specific content consumable by shared framework modules. Importantly, in our framework, external memory and cross-step reasoning history are not treated as unconditional parts of the base DeliveryBench observation. Retrieved memory is injected only when a memory module is enabled, and explicit reasoning history is included only when the corresponding history mechanism is turned on. This separation is important for fair modular comparison: it ensures that memory-augmented agents receive additional context because of their selected module design, rather than because the benchmark input itself differs across methods.

Action Representation. Finally, although DeliveryBench defines its own native action API, AGENTSPEC re-expresses these actions through a unified action abstraction. This preserves benchmark semantics at the execution level while allowing the same reasoning, memory, and reflection modules to operate across DeliveryBench, ALFRED, MiniGrid, and RoboTHOR under a common interface. As a result, differences in performance can be attributed more directly to module design rather than to environment-specific prompt engineering or ad hoc wrapper logic.

F.2 MINIGRID

MiniGridChevalier-Boisvert et al. (2023) is a symbolic grid-world benchmark for evaluating basic embodied decision-making. Unlike realistic 3D environments, MiniGrid uses compact grid layouts, discrete object types, and a small action space, making it useful for testing whether an agent can follow instructions, navigate, interact with objects, and solve simple compositional tasks under partial observability.

We construct an evaluation suite in MiniGrid by selecting ten tasks with diverse structures and difficulty levels, covering navigation, object interaction, and compositional reasoning. The selected tasks are as follows: (i) `empty_6x6_seed42`, (ii) `goto_object_6x6_n2_seed7`, (iii) `goto_door_5x5_seed123`, (iv) `fetch_5x5_n2_seed99`, (v) `simple_crossing_s9n1_seed42`, (vi) `lava_crossing_s9n1_seed123`, (vii) `multiroom_n2s4_seed42`, (viii) `lava_gap_s5_seed42`, (ix) `four_rooms_seed42`, and (x) `simple_crossing_s9n2_seed123`.

The agent receives first-person RGB observations together with a textual environment prompt as input. For all reasoning, memory, and reflection methods, we adopt the default configurations from our codebase without additional tuning. Each episode is executed with a maximum step budget of 100.

F.3 ALFRED

ALFREDShridhar et al. (2020) is a simulated household benchmark for long-horizon embodied task completion. It requires an agent to execute natural-language instructions of everyday activities in realistic indoor scenes.

We construct a lightweight evaluation suite from the ALFRED dataset by selecting seven representative tasks, each covering a distinct task type to ensure diversity in object manipulation, multi-object interaction, state-dependent reasoning, and long-horizon planning.

The selected tasks are: (i) place a bat on the bed, (ii) put two pillows on the sofa, (iii) put a heated apple in the fridge, (iv) cool bread and place it on the countertop, (v) clean a mug and place it on the coffee machine, (vi) examine a pencil under a desk lamp, and (vii) place a box containing keys on a chair.

These tasks collectively cover major ALFRED categories, including pick-and-place, multi-object manipulation, heating, cooling, cleaning, lighting-based inspection, and movable receptacles.

The agent receives first-person RGB observations along with a structured list of visible objects within a 1.5 m egocentric range to aid perception and grounding. The maximum step budget is 200. All methods adopt default configurations without additional tuning.

F.3.1 ROBOTHOR

RoboTHOR is a 3D indoor ObjectNav benchmark in which an agent navigates from a first-person viewpoint to a specified target object category. Each episode provides an initial agent pose and a target category, and the agent must navigate using a discrete action space:

`{MoveAhead, RotateLeft, RotateRight, LookUp, LookDown, Stop}`.

Following standard ObjectNav evaluation, the task uses stop-based success: the agent must explicitly issue `Stop`, and the target object must be visible within the configured success threshold. We report success rate (SR) and SPL, where SPL additionally accounts for path efficiency.

For our evaluation, we construct a compact but diverse RoboTHOR test suite by manually selecting 10 episodes from the validation set. The selected episodes cover different target object categories and navigation difficulties, ranging from targets located in the same room as the agent to targets requiring cross-room navigation and broader exploration. This design allows us to evaluate both local navigation behavior and longer-horizon exploration under a controlled number of episodes. We set the maximum episode length to 200 steps, the camera field of view to 90°, and the rotation angle for `RotateLeft` and `RotateRight` to 90°.

F.4 SUITES OF THE MAIN WORKFLOW STUDY

Table 16: Evaluation suites of the main workflow study. "Gate" is the four-task subset used to choose one of three round proposals before complete-suite scoring; both gate and complete-suite outcomes were returned to the designer; the unseen evaluation is the held-out suite of Table 2.

Benchmark	Complete suite	Gate	Step / lifecycle budget	Metric
DeliveryBench	`deliverybench_v0_small`, 2 tasks	none	one-hour lifecycle	hourly profit
ALFRED	`alfred_valid_unseen_small`, 10 tasks	4 tasks	$\min(200, 3\times\text{oracle})$ steps	SR (GC, GCSPL diagnostic)
MiniGrid	`minigrid_mixed_mid10`, 10 tasks	4 tasks	100 steps	SR (SPL diagnostic)
RoboTHOR	`robothor_objectnav_main`, 10 tasks	4 tasks	suite-defined	SR (SPL, PR diagnostic)

The executor receives each environment's declared observation and action schema through the common adapter: MiniGrid first-person partial RGB plus structured text fields (position, facing, the cell ahead, an on-goal flag), ALFRED and RoboTHOR first-person visual observations with their benchmark action sets, and DeliveryBench structured courier state with the native delivery actions. Settings of the controlled component study (Appendix C) are documented with that study and are not carried into Table 1, except that its MiniGrid suite and step cap are identical to the main study's.

F.5 SEARCH AND HELD-OUT SUITES OF THE PILOT CAMPAIGN

The search suite `babyai_goto` contains 36 episodes drawn from 23 BabyAI *GoTo* levels (GoToRedBall and its no-distractor and grey variants, GoToRedBlueBall, GoToLocal at several room sizes, GoToObj and maze variants, GoToDoor, GoToOpen, GoToImpUnlock, GoToSeq, GoTo) with per-episode step budgets between 100 and 300; MiniGrid additionally truncates at its own per-environment limit, and the digest reports the budget an episode actually had. The held-out suite `minigrid_navigation` contains 23 episodes from Empty, FourRooms, MultiRoom, SimpleCrossing, LavaCrossing, LavaGap, and Dynamic-Obstacles levels with budgets between 30 and 300. Observations are the first-person partial RGB render plus the adapter's text fields (agent position, facing, the cell in front, and an on-goal flag); the full grid is not exposed.

G SCOPE OF THE STUDY

Task families. The study covers embodied sequential decision-making on five benchmarks; the substrate is not specific to them.

Designer and initial library. Evolution quality depends on the coding agent and on the human-written library it starts from. The framework, adapters, evaluators, suites, gate, and library are human-written; the loop automates candidate-level implementation and revision, and infrastructure events during a campaign are listed in Appendix A with their timing and affected rounds.

Editing scope. The loop order, the adapters, and the evaluator are held fixed in this study, which keeps every candidate inspectable and comparable and bounds the reachable designs to the component space of Table 5; the framework itself can expose other components, including perception, to the same contracts.

What the numbers estimate. Each Table 1 entry is one execution; evolution scores are development-suite best-of-five values reported with every round score, and the frozen best harnesses of DeliveryBench and MiniGrid were re-executed three times and evaluated on a held-out MiniGrid suite (Table 2). The executed comparison varies feedback, search budget, library access, and editing scope together, so it characterizes complete construction workflows rather than isolating the contribution of typing; the substrate controls of Table 6 are the design for that isolation.