REA: A ROUTED EMBODIED AGENT RUNTIME FOR ASYNCHRONOUS BRAIN-CEREBELLUM CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Foundation models are becoming remarkably reliable at high-level reasoning, yet they remain too slow and costly to serve as fine-grained controllers. Conversely, fast policies like VLAs and whole-body controllers offer more efficient reflexes, but struggle to generalize. While pairing a deliberative brain (foundation models) with a reactive cerebellum (fast policies) seems natural for embodied systems, two obstacles remain: methods are built and scored on incompatible stacks, and their asynchronous coordination is difficult. For example, an agent that calls the VLA in the loop can stall the fast execution, while a VLA that calls the agent on errors tends to re-trigger fallbacks repeatedly or completely miss silent failures. To address this, we propose REA (Routed Embodied Agent), a unified runtime where a deliberative System 2, an embodied router System 1, and reactive System 0 skills cooperate under a single golden rule: no running skill waits on the brain. Instead of querying the brain on the critical path, REA employs a *route memory* that stores each skill's measured success ahead of time, a *boundary-triggered judge gate* that consults the brain only after a skill returns, and a shared-budget *escalation ladder* for bounded recovery. Evaluated on 3,000 held-out episodes across 60 LIBERO-Pro tasks, REA reaches 95.9% success and beats the best single-agent baseline in its own skill pool (94.7%) while reducing control steps by 25% and control time by 17%. Where skills fail silently, the judge gate adds 3.1 points over standard exception handling on unselected episodes, at 2.1× the time per success, and 15.3% on a failure-heavy stress subset. REA also carries over to BEHAVIOR-1K, RoboTwin 2.0 and a real Unitree G1 humanoid, solving the most episodes of the compared methods on all three. Our findings offer an insight for embodied design: an embodied brain is best utilized when its decisions are made ahead of time and queried only at natural behavioral boundaries, rather than wedged into the fixed control path.

1 INTRODUCTION

Foundation-model agents have become capable *embodied brains*: a coding agent can write a manipulation program that calls segmentation, grasp planning and inverse kinematics (Liang et al., 2023; Lu et al., 2026), and a language or vision–language model can decompose a command (Ahn et al., 2022; Huang et al., 2022; Driess et al., 2023) or tell whether a task succeeded (Driess et al., 2023; Du et al., 2023). These abilities are expensive: in our runs ASPIRE's repair loop (Lu et al., 2026) spent a median of 43 min per LIBERO task, and a Codex + GPT-6 pipeline consumed 89M input tokens to author programs for 60 LIBERO-Pro tasks. Nor can the brain simply act as the controller: in our probes, models emitting one-token or Cartesian IK actions solved 1 of 1,987 and 0 of 600 LIBERO-Pro episodes (Appendix Q, Figure 5). Vision–language–action models (VLAs) (Brohan et al., 2023; Kim et al., 2024; Octo Model Team et al., 2024; Black et al., 2025a) and whole-body controllers such as SONIC (Luo et al., 2026) act at a few to tens of hertz, far faster than an agent deliberates, but VLAs that exceed 90% on LIBERO collapse under LIBERO-Pro's perturbed objects, initial states and instructions (Zhou et al., 2025).

Embodied intelligence should therefore separate a deliberative *brain* from a reactive *cerebellum*, as recent dual-system robot models do (Bjorck et al., 2025; Figure AI, 2025; 2026; Shi et al., 2025; Gemini Robotics Team, 2025). Two obstacles stand in the way. First, **there is no common ground for comparing agents and VLAs**: agent- and VLA-based systems are each built on their own simulator, robot, observation format and success criterion, so they are rarely evaluated on the same

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.

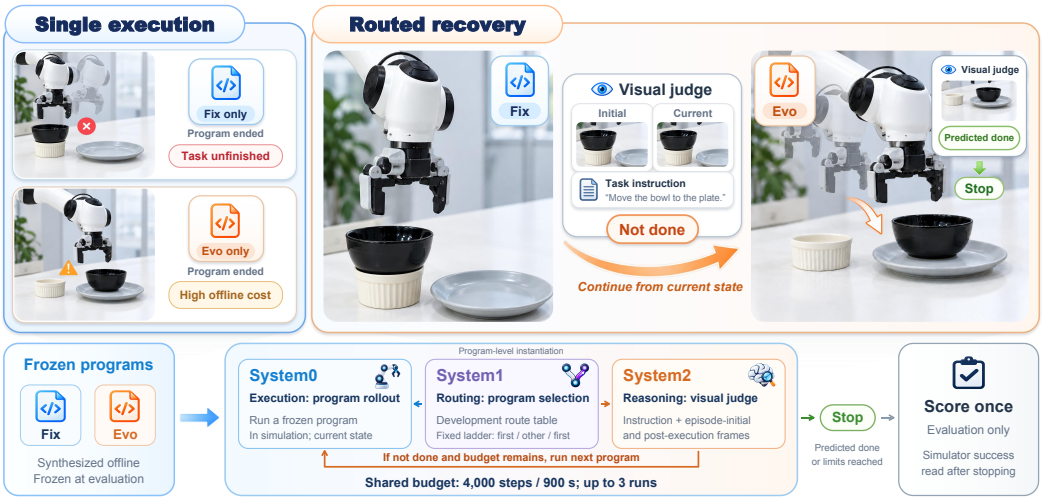


Figure 1: **Overview of REA with program skills.** Top left: a single skill may end with the task unfinished (Fix), and a stronger skill is expensive to synthesise (Evo). Top right: the System 2 visual judge compares the episode-initial and current frames with the instruction, answers *not done*, and the router dispatches the next skill from the current state. Bottom: System 0 runs frozen, offline-authored programs; System 1 selects them with a route table (*route memory*) and a first/other/first *escalation ladder*; System 2 is the visual *judge gate*. All runs share one budget; success is read once, after stopping, for evaluation only.

episodes. Second, **asynchronous coordination is hard** (Figure 2). When the agent drives and calls the fast policy, the fast loop waits on every agent decision: an agent advisor that the router waits for leaves the robot idle for 2.6 s per LIBERO-Pro episode, 20% of its control time (Figure 3). When the fast policy drives and calls the agent on errors, a persistent error re-triggers the brain at every recurrence, while most failures raise no error at all: an exception-triggered router escalated only 6 times in 590 LIBERO-Pro episodes although 166 failed. Blind re-execution is no remedy either, because re-running after a success can undo it.

We address both obstacles with REA, a *Routed Embodied Agent* runtime (Figure 1) meant to unify agent and VLA research across simulators and real robots: every component and backend implements one three-level contract. **System 2, the brain**, is an agent that authors skills offline and, online, only advises or judges; it never sits in the control loop. **System 1, the embodied router**, never blocks a running skill and decides which skill runs, when to consult the brain and when to stop. **System 0, the cerebellum**, holds fast skills (frozen programs, VLA policies or whole-body controllers) that return control to the router at segment or skill boundaries. Because every method is a configuration of one contract and is scored by one read of the benchmark's success predicate, REA doubles as an aligned harness: skills from different agents, routers with and without an online brain, and single-skill baselines run on identical tasks, seeds, initial states and budgets.

Our thesis is that the brain should work ahead of time or only when a skill returns, so that no running skill ever waits for it. This is what we mean by *asynchronous*: skills run at their own rate, and System 2 acts only between skills. The router does this with one module per coordination problem. **To keep the brain off the dispatch path**, REA contains a **route memory** of each skill's per-task success and cost on development episodes, so choosing a skill is a lookup. **To catch silent failures without re-firing on errors**, a **boundary-triggered judge gate** consults System 2 once per skill return: *not done* escalates, *done* stops and protects completed work. **To keep recovery bounded**, an **escalation ladder** dispatches the next skill from the current state under one shared step and time budget.

Results. On all 60 LIBERO-Pro goal, object and spatial tasks (50 held-out seeds each, 3,000 episodes per arm) REA with route memory and no online brain call (REA-M) reaches 95.9%, above each skill author in its pool run alone, including the strongest, Agents SDK + GPT-6 (94.7%; paired

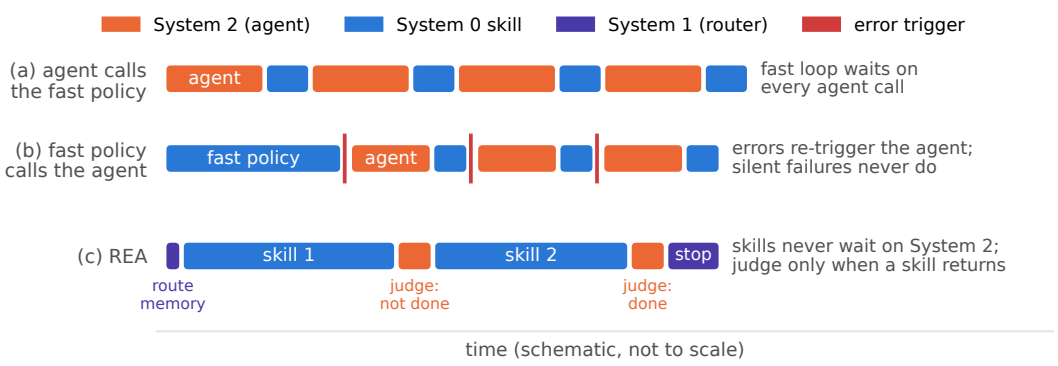


Figure 2: **Three ways to couple a slow brain with fast skills** (schematic). (a) An agent that calls the fast policy puts its latency on every step. (b) A fast policy that calls the agent on errors is interrupted by persistent errors and never by silent failures. (c) REA dispatches from route memory, lets each skill run uninterrupted at native rate, and consults the System 2 judge only when a skill returns, escalating from the current state.

86:51, p=0.004), within 1 pp of a hindsight per-task oracle, and with 25% fewer control steps per success. On the 37 tasks whose pool holds exactly these authors' programs the gain is +1.6 pp (p=0.001). It is also the fastest timed arm: 17% less control time than Agents SDK + GPT-6 and 25% less than an agent advisor that the router waits for. Where skills fail silently, REA-V (route memory + judge gate) improves ASPIRE and zero-shot GPT-6 skills by +8.6 and +7.0 pp (both $p<10^{-4}$); the judge gate itself adds +3.1 pp over an exception gate (p=0.022) at 2.1× the time per success, and +15.3 pp on a failure-heavy stress subset. The runtime also runs on BEHAVIOR-1K with only a new backend, on RoboTwin 2.0 and on a real Unitree G1 humanoid, solving the most episodes of the compared methods on all three (8 of 10 trials on the G1).

Contributions. (1) REA, a three-level embodied runtime whose contract admits agent, VLA and program skills under one evaluation protocol; we evaluate it on LIBERO-Pro, BEHAVIOR-1K and RoboTwin 2.0, on LIBERO-Pro with agent-written program skills (the strongest there, where VLAs collapse), and on a real Unitree G1 humanoid. (2) An embodied router whose three modules each target one failure mode of asynchronous brain–cerebellum coordination. (3) An aligned study over more than 30,000 held-out episodes with three findings: dispatching from route memory beats each skill author alone, matches an agent consulted at every dispatch and gives the fastest arm; on a strong library, moving the agent into other online roles adds waiting and no success; and where skills fail silently, a noisy judge gate adds +3.1 pp at twice the time per success.

2 Related Work

Embodied brains and fast policies. Large models serve as the deliberative layer of robot systems: SayCan (Ahn et al., 2022) grounds LLM plans in learned affordances, PaLM-E (Driess et al., 2023) embeds observations into a language model, and Inner Monologue (Huang et al., 2022) and SayPlan (Rana et al., 2023) re-plan from feedback. Code-as-policy methods let the agent write the controller: Code as Policies (Liang et al., 2023) and ProgPrompt (Singh et al., 2023) compose perception and motion primitives, Voyager (Wang et al., 2024) iterates on skill code with environment feedback (Eureka (Ma et al., 2024) does so for reward code), and ASPIRE (Lu et al., 2026) scales repair and evolutionary search to LIBERO; more generally, ReAct (Yao et al., 2023) interleaves reasoning with actions and Reflexion (Shinn et al., 2023) adds self-reflection across trials. At the other end, vision–language–action models such as RT-2 (Brohan et al., 2023), OpenVLA (Kim et al., 2024), Octo (Octo Model Team et al., 2024) and π_0 (Black et al., 2025a) act far faster than an agent deliberates but generalise within their training distribution (Zhou et al., 2025), and whole-body controllers such as SONIC (Luo et al., 2026) track motions commanded from above. REA keeps the agent as System 2, off the control loop and the dispatch path, and treats any fast policy as a System 0 skill.

Dual-system control. Following Kahneman (2011), GR00T N1 (Bjorck et al., 2025) and Helix (Figure AI, 2025) train a slow VLM with a fast action head that run at fixed, different rates, and Helix 02 (Figure AI, 2026) adds a learned 1 kHz whole-body "System 0"; Hi Robot (Shi et al., 2025) and Gemini Robotics 1.5 (Gemini Robotics Team, 2025) let a VLM planner command a VLA, and real-time chunking (Black et al., 2025b) overlaps a VLA's inference with its execution. Every level there is a learned model of one stack. REA keeps the level names but not their semantics: System 1 is a router that emits no actions, any policy can be a System 0 skill, and System 2 is an independently developed agent that no running skill waits for.

Routing, evaluation and completion judging. FrugalGPT (Chen et al., 2024) cascades LLM APIs, Lu et al. (2024) learn reward-guided routers, mixture-of-experts layers (Shazeer et al., 2017) gate per token, and Agent-as-a-Router (Zhou et al., 2026) routes each coding task using the verified outcomes of similar past tasks; REA's route memory brings outcome-based routing to embodied skills. Embodied Agent Interface (Li et al., 2024a) and SIMPLER (Li et al., 2024b) each evaluate one family of methods, LLM planners or VLAs, whereas REA runs brains, routers and skills of different families on identical episodes. VLMs serve as success detectors (Du et al., 2023) and reward models (Sontakke et al., 2023), LLM judges score model outputs (Zheng et al., 2023), and REFLECT (Liu et al., 2023b) turns detected failures into corrections; REA proposes no new detector, only where one sits: once per skill return, gating escalation under a shared budget and never used as the success criterion.

3 THE REA RUNTIME

REA answers one question: *how can a slow, general brain and fast, narrow skills cooperate without the brain ever blocking a running skill?*

3.1 THREE-LEVEL CONTRACT

An episode is a task t with instruction ℓ_t, an initial state indexed by seed s, a step horizon H and a wall-clock budget W.

System 0: skill execution. A skill p is any controller that can start from the current state and eventually returns, $(\Delta, \delta, e, \tau) \leftarrow \textsc{Run}(p;\, H-u,\, W-w)$, where u, w are the steps and seconds already spent, Δ, δ those of this run, e an exception or $\perp$ and τ a truncation flag. A skill may also yield at *segment* boundaries and be resumed: a VLA action chunk, a motion interval, or a program that pauses every 100 control steps with its stack preserved. Skills run their own closed loop at native rate. On LIBERO-Pro they are frozen Python programs written by coding agents; they call SAM 3 (Carion et al., 2026), ContactGraspNet (Sundermeyer et al., 2021), PyRoki inverse kinematics (Kim et al., 2025) and blocking motion primitives inside a sandbox that rejects identifiers (such as `getattr`) that could reach simulator state.

System 2: the brain, offline and online. Offline, $\textsc{Author}(t)$ produces candidate skills $\mathcal{P}_t$ (the "Frozen programs" box of Figure 1). Online, the brain offers two services: $\textsc{Judge}(f_0, f_k, \ell_t) \in \{\texttt{done}, \texttt{not_done}\}$ compares the episode-initial and current frames with the instruction, and $\textsc{Advise}(o)$ recommends a candidate from text inputs (used by REA-A at dispatch and by the blocking-advisor baseline). System 2 never reads simulator state, rewards or object poses, and never emits actions.

Backends implement RESET, RENDER, the skill API and a SCORE read once after the last run: LIBERO/LIBERO-Pro on MuJoCo/robosuite (Liu et al., 2023a; Todorov et al., 2012; Zhu et al., 2020), BEHAVIOR-1K on OmniGibson/Isaac Sim (Li et al., 2022), RoboTwin 2.0 on SAPIEN (Chen et al., 2026) and a Unitree G1, where SCORE is the operator's judgement.

3.2 THE EMBODIED ROUTER (SYSTEM 1)

The router owns every control-flow decision, with one module per coordination problem.

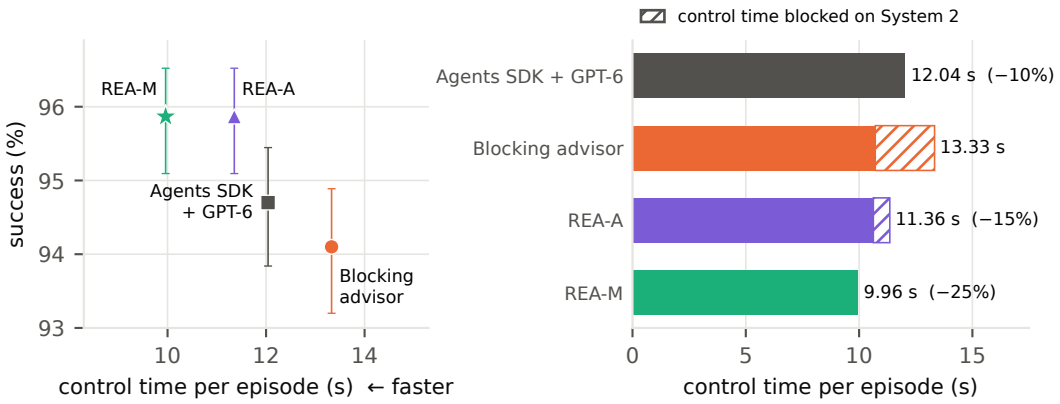


Figure 3: **Keeping the brain off the critical path** (LIBERO-Pro full set, 3,000 episodes per arm, one host; numbers in Table 3). Left: success (Wilson 95% intervals) against control time; up and to the left is better. Right: control time per episode (hatched: blocked on System 2; change vs. the blocking advisor in brackets). Every arm adds the same 2.6–2.7 s reset.

Route memory: dispatch without the brain. Before held-out evaluation, the router runs each candidate $p \in \mathcal{P}_t$ on n_p development episodes (seeds disjoint from held-out ones, seeded-random resets) and stores the success rate $\hat{r}_{t,p}$ and mean steps $\hat{c}_{t,p}$. The first skill is $\arg\max_p \hat{r}_{t,p}$, ties broken by lower $\hat{c}_{t,p}$: at test time dispatch is a lookup (the route table of Figure 1).

Boundary-triggered judge gate: consult the brain when a skill returns. Most failures are silent: RUN returns $e = \perp$ with the task unfinished, so an exception trigger never fires. REA therefore calls JUDGE once per skill return; `done` ends the episode and protects completed work, `not_done` escalates. A free *exception gate* escalates only on an exception or truncation and never calls System 2.

Escalation ladder with a shared budget. The ladder $L_t = [\text{first}(t), \text{other}(t), \text{first}(t)]$ (with three candidates, all three in route-memory order; one rung with one) launches each escalation from the current state, and every run and brain call draws on the same $(H - u, W - w)$, so an episode never exceeds a single-skill budget.

Configurations. REA-M: route memory, ladder, exception gate, no online System 2 call. REA-V: route memory, ladder, judge gate (episode loop in Algorithm 1, Appendix B). REA-A: an agent model that reads the route memory, the candidate sources, proprioception and the exception history replaces the lookup at dispatch and picks each exception-triggered escalation. A single-skill baseline is Algorithm 1 with K=0; blind retry uses the unconditional gate ALWAYS.

3.3 ALIGNED EVALUATION PROTOCOL

Because every method is a configuration of one runtime, arms are matched exactly: the same (task, seed) pairs and initial-state semantics, the same horizon and wall budget, the same frozen skill bytes wherever arms share a library, and one SCORE after the last run. Within a campaign on one host, each episode key keeps a static GPU lane. Route memories and judge thresholds use development episodes only and are frozen with their hashes recorded; Appendix H reports the audit.

4 EXPERIMENTS

We ask whether REA beats each skill author in its pool (Q1), is fast (Q2), helps any System 2 author (Q3), which module and which placement of System 2 cause the gains (Q4), and whether it carries over to other simulators (Q5).

Table 1: LIBERO-Pro full set (50 held-out seeds per task): REA against each skill author in its pool, run alone on the same episodes. *Aligned*: the 37 tasks whose pool is exactly the Codex, ASPIRE Fix and Agents SDK programs of the single-author rows (same program files); on the other 23 tasks a harness-generated or ASPIRE Evo program replaces the Codex or ASPIRE Fix one. The best author on development seeds 61–70 is Agents SDK (95.3%, vs. 75.0% Codex and 72.0% ASPIRE). Both routers use the exception gate (REA-M fails only 124 of 3,000 episodes); charts in Figure 7.

	Method	Calls/ep	All 60 (%)	Aligned 37 (%)	Steps/succ ↓
single author	Codex + GPT-6	0	74.4	81.2	1470
	ASPIRE Fix (59 / 36 tasks)	0	65.1	71.9	1566
	Agents SDK + GPT-6 (best dev.)	0	94.7	93.7	1343
	Random author per task	0	78.2	82.5	–
routed	Route memory only (K=0)	0	≥95.7	≥95.1	–
	REA-M (memory + ladder)	0	**95.9**	95.3	**1003**
	REA-A (agent at dispatch)	1.0	**95.9**	**95.5**	1110
ceiling	Hindsight per-task oracle	–	96.9	96.3	–

4.1 SETUP

Benchmarks. **LIBERO-Pro** (Zhou et al., 2025) perturbs LIBERO (Liu et al., 2023a) along object, position (*swap*), language, task and environment axes; we use its six goal, object and spatial suites under the *task* and *swap* perturbations, ten tasks each; held-out seed $s \in \{1, \ldots, 50\}$ uses official initial state $(s-1) \bmod 50$. The **full set** is all 60 tasks × 50 seeds (3,000 episodes per arm; the Long suite and the object, language and environment perturbations are outside our protocol). The **skill-author study** uses seeds 1–10 (600 episodes; 590 for ASPIRE arms, whose program for one task calls `getattr`, which the sandbox refuses). The **matched hard subset** stress-tests recovery on six tasks (five with two distinct skills; on SS-center the ASPIRE Evo winner equals Fix) with twelve seeds each, eight of them failures of the reference Fix skill (Table 5). On LIBERO-Pro every arm has a 4,000-step horizon and a 900 s budget; routed arms make at most three skill runs. BEHAVIOR-1K (Li et al., 2022) (R1Pro, Isaac Sim) and RoboTwin 2.0 (Chen et al., 2026) (bimanual, SAPIEN) and a real Unitree G1 humanoid test portability.

System 2 skill authors. ASPIRE Fix loop and Evo search (Lu et al., 2026); Codex + GPT-6 and Agents SDK + GPT-6, coding-agent pipelines authoring one program per task; and zero-shot GPT-6, two samples per task (Table 17). The *legacy library* holds a Codex or harness program per task and, for 59 tasks, an ASPIRE alternative; *+SDK* adds the Agents SDK programs (179 programs in total).

Router, advisor and judge. Route memories use development episodes only: seeds 51–60 on the matched subset and the skill-author study, and seeds 61–70 for the full set (1,190 episodes for the legacy programs, 600 for the SDK ones). REA-A's dispatch agent is text only; the *blocking advisor* of Q2 is a text-only agent that recommends a skill every 8 segments of 100 control steps while the router waits for it. The visual judge is `deepseek-v4-flash-vision-exp` on the matched subset and its successor `deepseek-flash` on LIBERO-Pro, with threshold and prompt frozen on 216 development labels before any held-out run (Appendix F). All arms of a comparison run on the same episodes: success differences use paired exact McNemar tests (McNemar, 1947) with paired 95% intervals, time differences task-clustered bootstrap intervals, and Appendix L reports Holm-adjusted p-values within each question. Models, versions and budgets are listed in Appendix N.

4.2 Q1: ROUTING BEATS EACH SKILL AUTHOR IN ITS POOL

Routing, not a larger pool, makes the gain. Table 1 compares REA with each author in its pool run alone on the same episodes. Choosing one author globally on development data selects Agents SDK (94.7%); choosing an author per task at random gives 78.2%. Route memory chooses per task on the same development seeds and reaches 95.9%, +1.2 pp over the best author (86:51, p=0.004, paired 95% CI $[+0.4, +1.9]$ pp). The comparison is exact on the 37 aligned tasks, where the pool holds only the single-author programs: there REA-M reaches 95.3% against 93.7% (54:25, p=0.001, CI $[+0.6, +2.5]$ pp), within 1.0 pp of the hindsight oracle over the whole pool; on the other 23 tasks

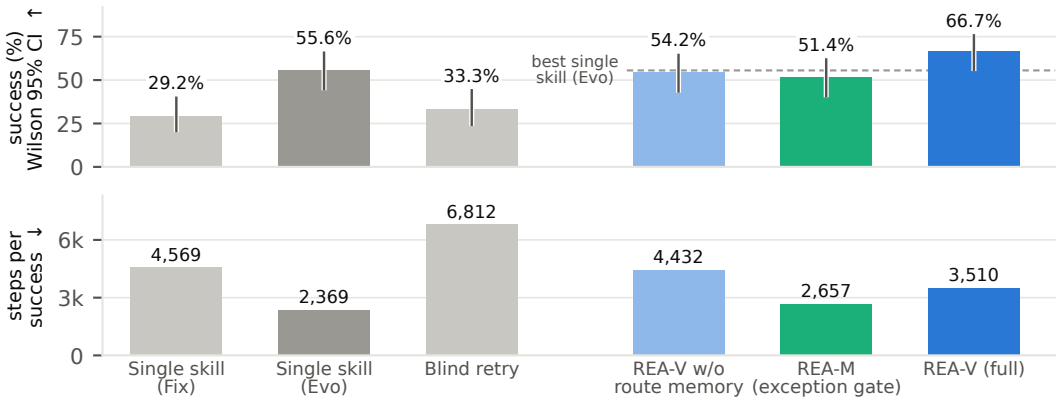


Figure 4: **Router ablation** on the matched hard subset (6 tasks $\times$ 12 seeds $=$ 72 episodes per arm; identical initial states, GPU lanes and skill bytes). Without route memory the judge ladder is Fix, Evo, Fix; blind retry re-runs Fix once. Numbers in Table 8, arm definitions in Table 6, per task in Figure 8.

it is on par with the best author (96.8 vs. 96.3%). It also costs less: 1,003 steps per success versus 1,343 (-25%), because each task goes to its most reliable development skill (the SDK program on only 14 of the 60 tasks), and those skills are shorter. **Per program, the selector uses less data than the authors.** Route memory uses 10 development seeds per program (1,790 episodes) and no model tokens, whereas each authoring pipeline spent 1,249–1,328 development episodes and 29–89M input tokens on its own library (Appendix K). *Route memory is as good as an agent at dispatch*: REA-A reaches the same 95.9% (26:26 paired) and, as Q2 shows, is slower.

4.3 Q2: The Brain Stays off the Critical Path

Figure 3 measures the designs of Figure 2 on one host and one skill library (numbers in Table 3). **Waiting for the brain is expensive.** An agent advisor that the router waits for, although consulted only 1.7 times per episode, leaves the control path idle 2.62 s per episode (20% of its control time), delays the first action by 1.5 s, and is slower than one SDK program (13.33 vs. 12.04 s) with no gain in success. Its advice also costs success, nearly all at the first dispatch: it overrides the route-memory skill in 1,236 episodes, on which that skill alone solves 1,168 and the advisor 1,124 (rest: 1,708 vs. 1,699). **Route memory removes the wait at its source.** REA-M has the lowest control time of the timed arms and ties REA-A for the highest success: -3.37 s (-25%, CI $[-31, -20]\%$) with $+1.8$ pp of success against the blocking advisor (90:37, $p<10^{-5}$), and -2.08 s (-17%, CI $[-23, -11]\%$) against Agents SDK + GPT-6 in its own campaign. The first gap is not hardware: both arms execute a control step in 10.3–10.4 ms outside System 2 waits. REA-A, whose agent reads the same route memory, matches its success but is 1.39 s slower per paired episode: 0.73 s waiting for its selection call, the rest because it starts a different program in 46% of episodes.

4.4 Q3: REA Improves Skills from Very Different Authors

On ASPIRE's Fix-loop skills (LIBERO-Pro seeds 1–10, 590 episodes; Table 12), REA-M alone lifts success from 66.3% to 71.9% ($+5.6$ pp, 49:16, $p<10^{-4}$) while *reducing* steps per success (1,467 vs. 1,557): it escalated on an exception only 6 times although 166 episodes failed, so nearly all of the gain is route memory running the Evo skill first where development favoured it. The judge gate (REA-V) adds a further $+3.1$ pp (37:19, $p=0.022$), for 74.9% in total, at $2.1\times$ the wall time per success (Figure 11). This paired contrast is the gate's net effect; how it splits into rescued failures, undone successes and replay variance is only approximate, because REA-M's own run stands in for REA-V's first run (Table 11). On zero-shot GPT-6 skills, a threefold weaker library, REA-V lifts success from 19.3% to 26.3% (50:8, $p<10^{-4}$); the comparison is unchanged on the 59 tasks shared with ASPIRE (50:8), so the task excluded by the sandbox does not drive it.

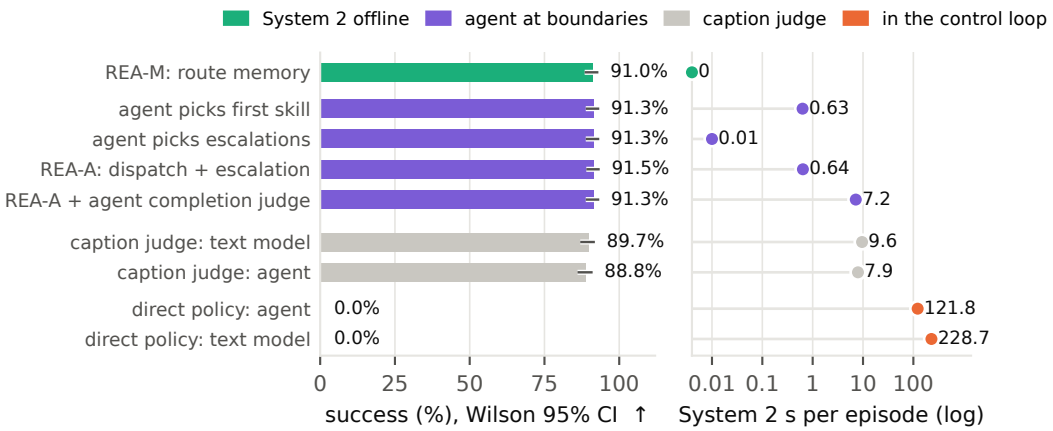


Figure 5: **Where should System 2 sit?** LIBERO-Pro seeds 1–10 (600 episodes per arm, one host, legacy library, exception gate unless stated). Left: success (Wilson 95% intervals). Right: seconds per episode spent in System 2 calls (agent, caption model or text judge), log scale. The direct policies act through 3 cm / 15° Cartesian IK steps and a gripper command, at most 80 decisions, from scene captions. Calls and paired counts in Table 4.

4.5 Q4: Where Do the Gains Come From?

The matched subset concentrates on episodes where single skills fail (Figure 4). The full router solves 48/72, +11.1 pp over the best single skill (10:2, p=0.039 unadjusted) and the only arm above it; the gate contrast below is the one that survives a Holm correction (Appendix L). **Both modules contribute.** Without route memory the judge must find the better skill by trial and error (39/72; 15:6, p=0.078). Replacing the judge gate by the exception gate costs 15.3 pp (12:1, p=0.003): of the twelve episodes only the full router solves, seven are failed first runs rescued by escalation and five are replay variance. **The comparison is at a matched escalation budget**: REA-M's cap K=1 never binds, because all three of its escalations ran to the 4,000-step horizon, so with K=2 it would behave identically on every episode. The subset is selected for Fix failures; on unselected seeds the same gate contrast is +3.1 pp (Q3). **Uncoordinated recovery does not help**: blind retry barely beats its own first skill (24 vs. 21; 12:9, p=0.66) and turns 4/12 successes into 1/12 on SS-between, where a second grasp disturbs a placed object; stopping on `done` and escalating to the other skill avoids this (11/12 even without route memory).

Where should System 2 sit? Figure 5 moves the agent through every online role on one library and host. Asking it at dispatch, at recovery or both changes success by +0.3 to +0.5 pp (all $p \geq 0.51$) while adding a selection call. A completion judge that reads scene captions, whether the agent or a text model, costs 1.3–2.2 pp and spends 7.9–9.6 s per episode in System 2 calls, since on this strong library the exception gate suffices; as a direct Cartesian policy it solves none of 600 episodes, a second failing interface after the one-token probe (Appendix Q). The brain pays off offline and, where skills fail silently, as a visual judge.

Why can a noisy judge help? Its final verdicts are barely more accurate than always answering `done` (0.72 vs. 0.67 matched) or less (0.67 vs. 0.75 LIBERO-Pro), but their `done` decisions are precise (0.83, 0.905), and re-running a completed task usually keeps it completed: by the proxy attribution of Table 11, re-runs convert 24% (19%) of failed first runs and undo 5% (9%) of successful ones. Measured on final verdicts only, this is a consistent explanation, not a tested mechanism; the +15.3 pp of this subset is a stress test, and +3.1 pp on unselected seeds (Q3) is the estimate to expect.

4.6 Q5: Other Simulators and a Real Robot

On BEHAVIOR-1K (R1Pro, Isaac Sim), which needed only a new backend, the full router solves 14 of 20 held-out episodes, in less time per episode than the online Agents SDK agent (Figure 6; times in Table 14). RoboTwin 2.0 has no skill library, so a fast reflex model is the System 0 skill and the

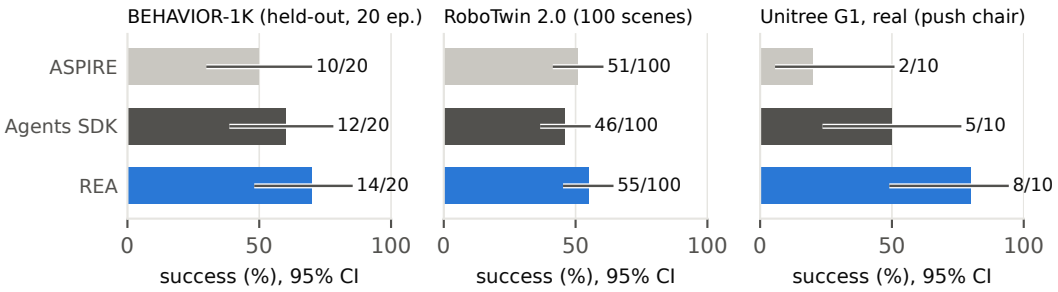


Figure 6: **Portability.** BEHAVIOR-1K: held-out instances 1–10 of its two ASPIRE tasks, with the full router (agent dispatch, judge gate, ladder); ASPIRE is its human-written programs and Agents SDK an online agent. RoboTwin 2.0: the same 100 frozen scenes (50 tasks, one clean and one randomized each) for every method, `gpt-6-astra`, at most 40 decisions per episode; there a reflex model is the System 0 skill. Unitree G1: real walk-and-push-chair trials, operator-judged (Codex, run separately, in Table 18). Details in Appendices J and O.

router hands single decisions to a deliberate model; it solves the most scenes, 55 of 100, against 51 for ASPIRE and 46 for the Agents SDK agent. At these sample sizes the margins on both simulators are within run-to-run variation (Appendix J). On a Unitree G1 humanoid that must walk to a chair and push it, with the same model and System 0 skills for every method and success judged on site, REA succeeds in 8 of 10 trials, against 5 for the Agents SDK agent, 4 for Codex and 2 for ASPIRE (p=0.023; Table 18).

5 Discussion

Use the brain ahead of time or at skill returns, never inside a running skill. Dispatching from route memory beats each skill author alone and is as accurate as asking an agent at every dispatch, at no online cost (Table 1, Figure 5); consulted in the loop, the brain adds waiting (Figure 3); consulted only when a skill returns, a noisy judge adds success where skills fail silently, at extra time (Table 12, Figure 4). The brain still sets the ceiling (skill authors differ by 20–47 pp), and the router decides how much of it is reached. The gate should be chosen per library on development episodes: the free exception gate suffices when skills rarely fail, the judge gate pays for its calls when they fail silently.

Scope. On LIBERO-Pro our System 0 skills are agent-written programs, chosen because VLAs collapse there (Zhou et al., 2025) while programs reach 65–96%; success is measured in simulation except for one real-robot task with 10 trials per method, and VLA, whole-body and broader real-robot skills, which use the same RUN interface, are the natural next step. Route memory is fitted per task, so held-out tests change seeds and initial states, not the task set; for a new task without development statistics, REA falls back to the author that is best on development data overall (94.7% here).

6 Conclusion

REA separates an embodied system into a System 2 brain, an embodied router and fast System 0 skills under one aligned protocol, and never makes a running skill wait for the brain. On 60 LIBERO-Pro tasks it beats each skill author in its pool and is the fastest timed pipeline without any online brain call. Its gains come from the router rather than a larger pool: route memory dispatches as well as an online agent at no model cost, and the boundary-triggered judge helps where skills fail silently. The same interfaces carry over to BEHAVIOR-1K, RoboTwin 2.0 and a real Unitree G1, where REA solves the most episodes. Because System 0 is reached only through RUN, VLA and whole-body skills plug in unchanged, which is the natural next test. We hope REA becomes a shared platform for developing and comparing embodied brains and fast skills.

AI USE STATEMENT

Large language models are part of the studied system: GPT-6 (through the Codex CLI and the OpenAI Agents SDK) and zero-shot prompting author the System 0 skills, and text and vision models act as dispatch agents, advisors and completion judges, as specified in Appendix N. In preparing the paper, the authors used LLM-based assistants to help review the manuscript. The authors designed the method and experiments, verified every number, figure and citation, and take full responsibility for the content.

ETHICS STATEMENT

This work involves no human subjects, personal data or crowd-sourced annotation; the 216 judge development labels were produced by the authors from simulator frames. Simulated experiments use public benchmarks (LIBERO-Pro, BEHAVIOR-1K, RoboTwin 2.0) under their licenses. The real-robot trials (Appendices O and P) ran a Unitree G1 secured by a gantry at low walking speed, with an operator present and a stop-and-stand recovery after every trial. Because REA executes agent-written programs on a robot interface, it runs every skill in a sandbox that refuses reflective calls, and the shared step and wall-clock budget bounds every episode (Section 4.1); deployment on real hardware would additionally require independent safety limits below the skill interface. We declare no conflicts of interest.

REPRODUCIBILITY STATEMENT

Every number in this paper is computed from per-episode records (plans, run logs, frames and one final score per episode) that we will release with the runtime, the frozen skill libraries, the route memories, the frozen judge configuration and its 216 development labels. Section 4.1 and Appendices C and H give the task lists, seeds, budgets and arm configurations; Appendix A details the timing definitions.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, 2022.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, et al. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems (RSS)*, 2025a. doi: 10.15607/RSS.2025.XXI.010.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025b.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning (CoRL)*, 2023.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris, Chaitanya Ryali, Kalyan Vasudev Alwala, et al. SAM 3: Segment anything with concepts. In *International Conference on Learning Representations (ICLR)*, 2026.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research (TMLR)*, 2024.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, et al. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. In *International Conference on Machine Learning (ICML)*, 2026.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. PaLM-E: An embodied multimodal language model. In *International Conference on Machine Learning (ICML)*, 2023.

Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-language models as success detectors. In *Conference on Lifelong Learning Agents (CoLLAs)*, 2023.

Figure AI. Helix: A vision-language-action model for generalist humanoid control. `https://www.figure.ai/news/helix`, 2025.

Figure AI. Introducing Helix 02: Full-body autonomy. `https://www.figure.ai/news/helix-02`, 2026.

Gemini Robotics Team. Gemini Robotics 1.5: Pushing the frontier of generalist robots with advanced embodied reasoning, thinking, and motion transfer. *arXiv preprint arXiv:2510.03342*, 2025.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning (CoRL)*, 2022.

Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

Chung Min Kim, Brent Yi, Hongsuk Choi, Yi Ma, Ken Goldberg, and Angjoo Kanazawa. PyRoki: A modular toolkit for robot kinematic optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Pannag Sanketi, Quan Vuong, et al. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning (CoRL)*, 2024.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, et al. BEHAVIOR-1K: A benchmark for embodied AI with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning (CoRL)*, 2022.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, et al. Embodied agent interface: Benchmarking LLMs for embodied decision making. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024a.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, et al. Evaluating real-world robot manipulation policies in simulation. In *Conference on Robot Learning (CoRL)*, 2024b.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023a.

Zeyi Liu, Arpit Bahety, and Shuran Song. REFLECT: Summarizing robot experiences for failure explanation and correction. In *Conference on Robot Learning (CoRL)*, 2023b.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. Routing to the expert: Efficient reward-guided ensemble of large language models. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2024.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Yuke Zhu, Linxi Fan, and Guanzhi Wang. ASPIRE: Agentic /skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026.

Zhengyi Luo, Ye Yuan, Tingwu Wang, Chenran Li, Fernando Castañeda, Sirui Chen, et al. SONIC: Supersizing motion tracking for natural humanoid whole-body control. *Science Robotics*, 11(117): eaed4592, 2026. doi: 10.1126/scirobotics.aed4592.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems (RSS)*, 2024.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Sünderhauf. SayPlan: Grounding large language models using 3D scene graphs for scalable robot task planning. In *Conference on Robot Learning (CoRL)*, 2023.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, et al. Hi Robot: Open-ended instruction following with hierarchical vision-language-action models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 54919–54933, 2025.

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

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Sumedh A. Sontakke, Jesse Zhang, Sébastien M. R. Arnold, Karl Pertsch, Erdem Bıyık, Dorsa Sadigh, Chelsea Finn, and Laurent Itti. RoboCLIP: One demonstration is enough to learn robot policies. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. Contact-GraspNet: Efficient 6-DoF grasp generation in cluttered scenes. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

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 (TMLR)*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

Pengfei Zhou, Zhiwei Tang, Yixing Ma, Jiasheng Tang, Yizeng Han, Zhenglin Wan, Fanqing Meng, Wei Wang, Bohan Zhuang, Wangbo Zhao, and Yang You. Agent-as-a-Router: Agentic model routing for coding tasks. *arXiv preprint arXiv:2606.22902*, 2026.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. LIBERO-PRO: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Kevin Lin, Abhiram Maddukuri, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

A Full-Set Charts and Speed Details

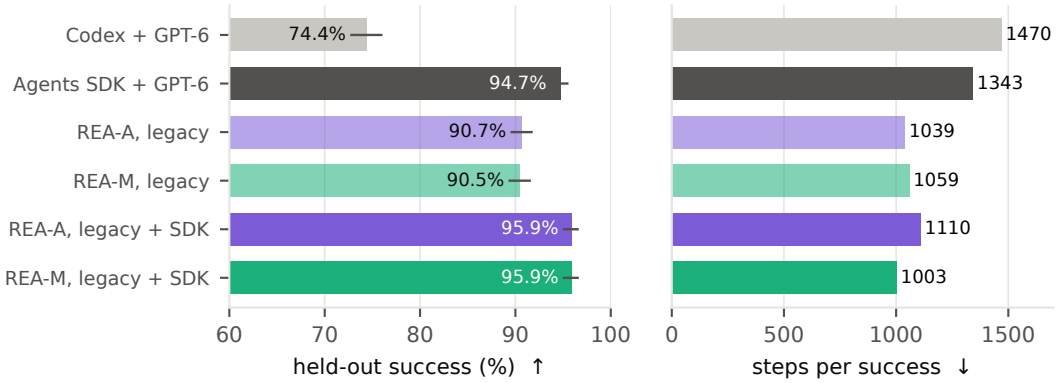


Figure 7: LIBERO-Pro full set: success (left; Wilson 95% intervals, axis from 60%) and steps per success (right) for arms of Table 1 (*legacy + SDK*: the 179-program library) and for both routers on the legacy library alone (*legacy*).

Timing definitions. *Control time* is the time from the end of the environment reset to the end of the episode; it includes every skill run and every System 2 call the control path waits for, and is the quantity charged against the 900 s budget. *Episode time* adds the reset (2.6–2.7 s on average in every arm). *Wait* is the time the control path is blocked on System 2. All timed arms of Table 3 ran on one host. The blocking advisor ran as its own campaign, with up to four workers sharing one GPU and private perception services; the Agents SDK, REA-M and REA-A arms ran in an earlier campaign with six workers on six GPUs. Outside System 2 waits the two campaigns execute at the same rate: 10.3 ms per control step for the blocking advisor, 10.4 ms for REA-M, 10.0 ms for REA-A and 9.5 ms for Agents SDK + GPT-6, so REA-M's 3.37 s advantage over the blocking advisor is its 2.62 s wait plus 0.75 s from 83 fewer control steps per episode. Intervals in Section 4.3 resample the 60 tasks (4,000 bootstrap draws). Codex + GPT-6, ASPIRE Fix and the legacy-library routers of Figure 7 ran on a different host (53.5, 44.3 and 55.9–56.0 s control time per episode) and are not timed. The blocking-advisor campaign logged 15 System 2 errors; in 12 episodes no advice completed and the route-memory skill ran alone. These episodes are kept.

Table 2: Per-suite success (of 500) and mean control time (s) on the full set. REA-M is faster than the blocking advisor in every suite and at least as accurate in every suite.

	Blocking advisor		REA-M	
Suite	Succ	Control	Succ	Control
goal_swap	456	14.30	470	11.32
goal_task	435	14.19	442	9.94
object_swap	474	10.06	499	7.68
object_task	481	12.36	486	7.43
spatial_swap	491	14.75	491	11.02
spatial_task	486	14.34	488	12.38

Table 3: Speed on the LIBERO-Pro full set, ordered by where System 2 runs (3,000 episodes per arm on one host; the routed arms share one 179-program library). Control: control time per episode, excluding reset; Δ: paired change in control time and b:c: episodes solved only by the row's arm : only by the blocking advisor; Wait: time per episode the control path is blocked on System 2. The blocking advisor ran in its own campaign on the same host, at the same execution rate outside System 2 waits. Codex + GPT-6 ran on another host and is not timed.

Method	System 2 runs	Calls/ep	Succ. (%) ↑	b:c	Control (s) ↓	Δ	Wait (s) ↓
Agents SDK + GPT-6	offline	0	94.7	109:91	12.04	-10%	0
Blocking advisor	in the loop	1.71	94.1	–	13.33	–	2.62
REA-A	at dispatch	1.00	**95.9**	97:44	11.36	-15%	0.73
REA-M	offline	0	**95.9**	90:37	**9.96**	$\mathbf{-25\%}$	0

Table 4: Where should System 2 sit (Figure 5)? LIBERO-Pro seeds 1–10 (600 episodes per arm, one host, legacy library, exception gate unless stated). b:c against REA-M; S2 time: seconds per episode spent in System 2 calls (agent, caption model or text judge). The per-step policies act through 3 cm / 15° Cartesian IK steps and a gripper command, at most 80 decisions, from scene captions.

System 2 placement	Configuration	Calls/ep	S2 time (s)	Success (%)	b:c
offline	REA-M (route memory)	0	0	91.0	–
at dispatch	agent picks first skill	0.98	0.63	91.3	6:4
at recovery	agent picks escalations	0.01	0.01	91.3	4:2
dispatch + recovery	REA-A	0.99	0.64	**91.5**	6:3
+ completion	REA-A + agent judge	4.18	7.18	91.3	12:10
caption judge	text model	3.13	9.57	89.7	11:19
	agent	3.23	7.94	88.8	9:22
direct policy	agent	52.9	121.8	0.0	0:546
	text model	90.1	228.7	0.0	0:546

B Episode Loop

Algorithm 1 gives the episode loop of the judge- and exception-gated configurations (Section 3.2).

Algorithm 1: One REA episode with the judge or exception gate. Nothing before the last line reads the success predicate, rewards or object poses; the verdict after the last run is only logged.

Input: task t, seed s, route memory $\mathcal{M}$, judge J, gate $g \in \{\text{JUDGE}, \text{EXCEPTION}, \text{ALWAYS}\}$, escalations K, budgets H, W

$L \leftarrow \mathcal{M}.\text{ladder}(t)$; RESET($s$); $f_0 \leftarrow$ RENDER(); $u, w \leftarrow 0$ ▷ System 1 lookup

for $k \leftarrow 0$ **to** K **do**

  $(\Delta, \delta, e, \tau) \leftarrow$ RUN($L[\min(k, |L|-1)]$; $H-u$, $W-w$); $u \mathrel{+}= \Delta$; $w \mathrel{+}= \delta$ ▷ System 0

  if $g =$ JUDGE **then** $(v, \delta_J) \leftarrow J(f_0, \text{RENDER}(), \ell_t)$; $w \mathrel{+}= \delta_J$ ▷ System 2

  if $k = K$ ***or*** $u \geq H$ ***or*** $w \geq W$ **then break**

  if $(g =$ JUDGE ***and*** $v =$ `done`$)$ ***or*** $(g =$ EXCEPTION ***and*** $e = \perp$ ***and not*** $\tau)$ **then break**

end

return SCORE() ▷ read once, evaluation only

C MATCHED HARD SUBSET: TASKS AND ARMS

Table 5: Tasks of the matched hard subset. Five tasks have two distinct candidate skills (ASPIRE Fix and Evo); on SS-center the Evo winner is byte-identical to Fix, so its two columns are two replays of the same program and differ only by replay variance. Eight of the twelve seeds per task were failures of the reference Fix skill.

Suite	Task (abbreviated)	Seeds	Fix/12	Evo/12
libero_goal_swap	Open drawer, put bowl (GS-drawer)	31–39, 42, 44, 47	3	11
libero_goal_task	Open drawer, put cream cheese (GT-drawer)	31–42	4	5
libero_object_task	Pick up butter, place in basket (OT-butter)	31–39, 46, 47, 50	2	2
libero_spatial_swap	Bowl between plate and ramekin (SS-between)	8, 14, 31–40	4	12
libero_spatial_swap	Bowl from table center (SS-center)	31–41, 43	3	7
libero_spatial_swap	Bowl on ramekin (SS-ramekin)	31–40, 43, 44	5	3

Table 6: Arm configurations of the router ablation (Table 8). All arms run the episode loop of Algorithm 1; blind retry uses the unconditional gate ALWAYS. Exception escalations were rare (3 in 72 episodes) and all ran to the horizon, so the cap $K{=}1$ of REA-M never binds.

Arm	First skill	Ladder	Gate	Max escalations K
Single skill (Fix / Evo)	Fix / Evo	[Fix] / [Evo]	–	0
Blind retry	Fix	[Fix, Fix]	always	1
REA-V w/o route memory	Fix	[Fix, Evo, Fix]	judge	2
REA-M	route memory	route memory	exception	1
REA-V (full)	**route memory**	**route memory**	**judge**	**2**

D PER-TASK RESULTS ON THE MATCHED HARD SUBSET

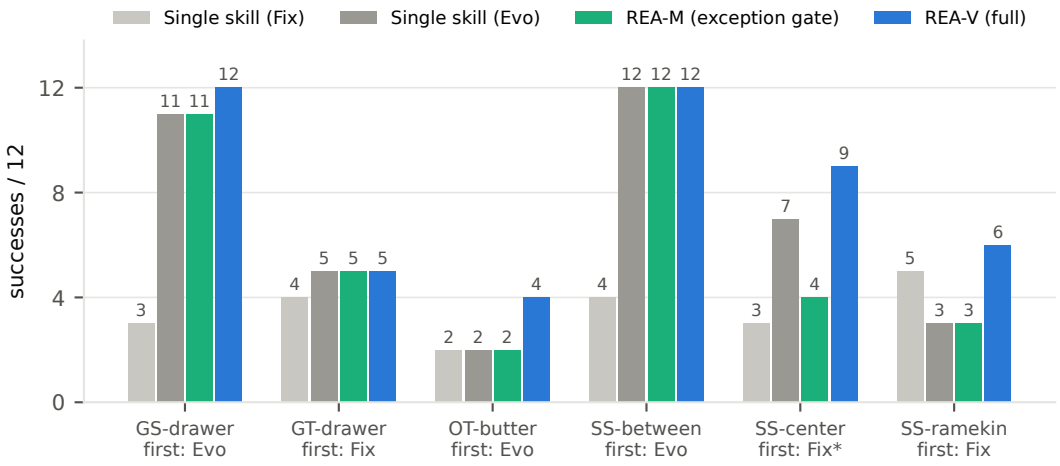


Figure 8: Successes out of 12 per task; the skill that route memory dispatches first is shown under each task (* SS-center has a single distinct skill). REA-V is best or tied-best on every task.

Table 7: Full per-task statistics (steps include the 10 settle steps; escalation steps are the steps of every run after the first). On SS-center, Fix and Evo are the same program.

Task	Arm	Succ/12	Steps/ep	Steps/succ	Judge calls/ep	Escalation steps/ep
GS-drawer	Single skill (Fix)	3	2312	9246	0	0
	Single skill (Evo)	11	2138	2332	0	0
	REA-V w/o route memory	7	3581	6138	1.92	1267
	Blind retry	6	3987	7974	0	1684
	REA-V (full)	12	3846	3846	1.92	1682
	REA-M (exception gate)	11	2138	2332	0	0
GT-drawer	Single skill (Fix)	4	2579	7738	0	0
	Single skill (Evo)	5	2630	6313	0	0
	REA-V w/o route memory	5	4000	9600	2.08	1310
	Blind retry	5	3958	9499	0	1310
	REA-V (full)	5	4000	9600	2.08	1310
	REA-M (exception gate)	5	2865	6876	0	282
OT-butter	Single skill (Fix)	2	1200	7197	0	0
	Single skill (Evo)	2	1213	7279	0	0
	REA-V w/o route memory	3	3230	12,920	2.92	2037
	Blind retry	3	2175	8700	0	967
	REA-V (full)	4	3092	9276	2.92	1879
	REA-M (exception gate)	2	1213	7279	0	0
SS-between	Single skill (Fix)	4	591	1774	0	0
	Single skill (Evo)	12	647	647	0	0
	REA-V w/o route memory	11	1066	1163	1.83	462
	Blind retry	1	1136	13,628	0	512
	REA-V (full)	12	647	647	1.00	0
	REA-M (exception gate)	12	647	647	0	0
SS-center	Single skill (Fix)	3	632	2529	0	0
	Single skill (Evo)	7	599	1026	0	0
	REA-V w/o route memory	9	1066	1422	1.92	497
	Blind retry	6	1178	2356	0	556
	REA-V (full)	9	1076	1435	1.75	444
	REA-M (exception gate)	4	637	1911	0	0
SS-ramekin	Single skill (Fix)	5	682	1638	0	0
	Single skill (Evo)	3	669	2677	0	0
	REA-V w/o route memory	4	1460	4379	2.25	785
	Blind retry	3	1191	4763	0	545
	REA-V (full)	6	1380	2760	2.17	697
	REA-M (exception gate)	3	692	2769	0	0

Table 8: Router ablation on the matched hard subset (Figure 4; 6 tasks $\times$ 12 seeds $= 72$ episodes per arm; Wilson intervals; identical initial states, GPU lanes and skill bytes). Without route memory the judge ladder is Fix, Evo, Fix, and blind retry re-runs Fix once; per-task results in Table 7.

Arm	Memory	Gate	Succ/72	Succ. (%) ↑	95% CI	Steps/succ ↓
Single skill (Fix)	–	–	21	29.2	[19.9, 40.5]	4569
Single skill (Evo)	–	–	40	55.6	[44.1, 66.5]	2369
Blind retry	–	always	24	33.3	[23.5, 44.8]	6812
REA-V w/o route memory	–	judge	39	54.2	[42.7, 65.2]	4432
REA-M	✓	exception	37	51.4	[40.1, 62.6]	2657
REA-V (full)	✓	judge	**48**	**66.7**	[55.2, 76.5]	3510

E ROUTE MEMORY

Table 9 lists the route memory fitted for the matched subset.

Table 9: Route memory of the matched subset, fitted on development seeds 51–60 (seeded-random resets). Evo is dispatched first exactly where development shows it to be clearly stronger. On SS-center the Evo program is byte-identical to Fix, so both development arms count as Fix evidence (6/20) and the ladder collapses to [Fix, Fix].

Task	Suite	Fix dev	Evo dev	First	Ladder
GS-drawer	goal_swap	1/10	10/10	Evo (higher rate)	Evo, Fix, Evo
GT-drawer	goal_task	1/10	1/10	Fix (tie; fewer steps)	Fix, Evo, Fix
OT-butter	object_task	0/10	3/10	Evo (higher rate)	Evo, Fix, Evo
SS-between	spatial_swap	1/10	10/10	Evo (higher rate)	Evo, Fix, Evo
SS-center	spatial_swap	6/20	= Fix	Fix (single skill)	Fix, Fix
SS-ramekin	spatial_swap	4/10	2/10	Fix (higher rate)	Fix, Evo, Fix

F VISUAL-JUDGE CALIBRATION

The visual judge is calibrated on development data only: 216 labelled episodes (seeds 51–62, 56 successes and 160 failures) whose initial and final 800×512 agentview frames were saved, with labels from the final score. We sweep the prompt style and whether the initial frame is shown, at thresholds 0.5–0.9, and select by accuracy, then precision (Figure 9 and Table 10). The successor model used on LIBERO-Pro scores 0.782 on the same 216 labels (0.792 for the frozen snapshot).

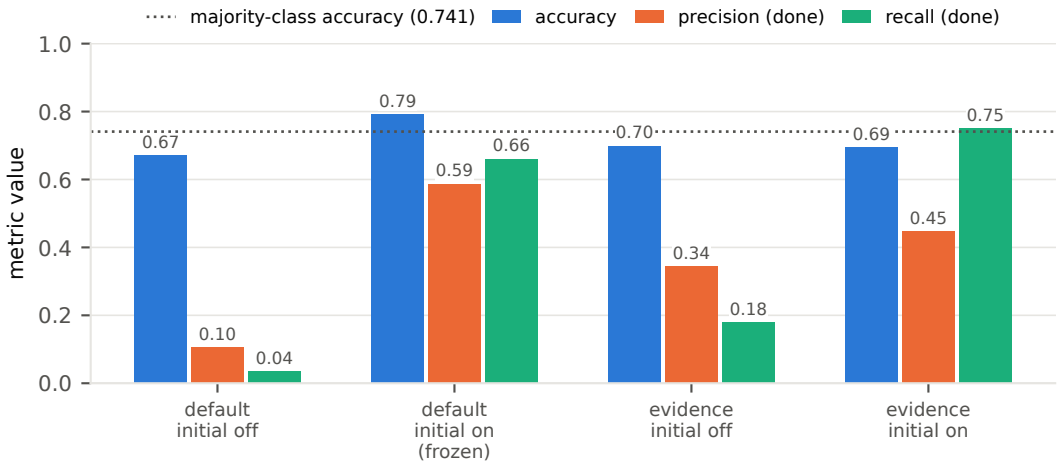


Figure 9: Judge calibration on development labels at threshold 0.6 (`done` is the positive class). With the default prompt, showing the episode-initial frame lifts the judge above the majority-class baseline; without it, precision and recall collapse.

Table 10: Development calibration sweep (`deepseek-v4-flash-vision-exp`, threshold 0.6; the result is flat over 0.5–0.9). The frozen configuration is in bold.

Prompt	Initial frame	Accuracy	Precision	Recall	Latency (s)	Cost/call (USD)
default	off	0.671	0.105	0.036	1.60	0.000247
default	**on**	**0.792**	**0.587**	0.661	1.60	0.000390
evidence	off	0.699	0.345	0.179	1.93	0.000353
evidence	on	0.694	0.447	0.750	2.01	0.000527

G MECHANISM OF THE JUDGE GATE

Table 11 attributes the episodes of the full router, using REA-M (same first skill, seed and lane) as a proxy for the outcome of the first run. Held-out confusion matrices of the final verdict against the

final score are TP 35 / FP 7 / FN 13 / TN 17 on the matched subset (done-precision 0.83) and TP 276 / FP 29 / FN 166 / TN 119 on LIBERO-Pro (done-precision 0.905). These count only the verdict after the last run, not every verdict that triggered an escalation.

Table 11: Episode attribution of the full router (proxy for the first-run outcome: REA-M). Re-run episodes had failed first runs in 29 of 50 cases on the matched subset but in only 145 of 362 on LIBERO-Pro; the gate converts 24% (19%) of the failures and undoes 5% (9%) of the successes it re-runs. Because REA-M only stands in for the first run, the split of the paired wins into rescues and replay variance ($12 = 7 + 5$; $37 = 28 + 9$) is approximate, and some of the undone successes may be replay variance in the other direction.

Attribution (proxy = REA-M outcome)	Matched (72)	LIBERO-Pro (590)
Rescued: first run failed → escalated → success	7	28
Not rescued: first run failed → escalated → still failed	22	117
Survived: first run succeeded → judged not done → re-run → success	20	198
Harmed: first run succeeded → re-run → failed	1	19
Forfeited: first run failed → judged done → stopped	1	12
Kept: first run succeeded and judged done	16	207
Replay variance: proxy failed but the router's own first run succeeded	5	9
Conversion of failures, rescued / (rescued + not rescued)	24%	19%
Destruction of successes, harmed / (harmed + survived)	5%	9%
Paired result vs. REA-M	$+12/-1$, p=0.003	$+37/-19$, p=0.022

H PROTOCOL AUDIT

For the matched-subset campaign (Table 8) we verify after the fact, from the stored plans and per-episode records, that all arms share:

- identical task hashes in every run plan and input manifest;
- identical GPU lane per (task, seed) key across arms (0 mismatches);
- held-out phase and official initial-state semantics in every plan and every per-episode record;
- the frozen judge configuration hash and the development-only route-memory hash (seeded-random development resets) in every plan that uses them;
- exactly one call to the final score per episode, and no read of the success predicate, rewards or object poses on any skill, router or judge path;
- identical skill-program hashes per task across arms;
- the 900 s wall budget and 4,000-step horizon (the wall budget was never binding; minimum remaining 723 s).

The LIBERO-Pro campaigns share task identities, seeds, initial-state semantics, horizon, wall budget and the single final score. Their judge is the successor model `deepseek-flash`, so their plans record the judge configuration but not the frozen hash; Codex + GPT-6, ASPIRE Fix and the legacy-library routers ran on a different host from the other full-set arms, so GPU lanes are not matched across hosts and times are compared only among same-host arms (Appendix A), and arms that compare skill authors or libraries use different skill bytes by design.

I LIBERO-PRO SKILL-AUTHOR STUDY: FULL TABLE

Table 12: LIBERO-Pro seeds 1–10: REA on skills from two System 2 authors (Section 4.4). $b{:}c$ = episodes solved only by the REA arm : only by the skills-only arm (paired exact McNemar). The zero-shot baseline is the better of two samples; ASPIRE rows cover 590 episodes; best per group in bold.

Arm	Succ	Succ. (%) ↑	Steps/succ ↓	$b{:}c$	p
GPT-6 zero-shot skills	116/600	19.3	**4299**	–	–
+ REA-V (memory + judge gate)	**158/600**	**26.3**	6023	50:8	$< 10^{-4}$
ASPIRE Fix-loop skills	391/590	66.3	1557	–	–
+ REA-M (memory + exception gate)	424/590	71.9	**1467**	49:16	$< 10^{-4}$
+ REA-V (memory + judge gate)	**442/590**	**74.9**	2513	73:22	$< 10^{-4}$

Table 13: LIBERO-Pro, held-out seeds 1–10, with latency columns (seconds; wall includes judge latency). ASPIRE rows cover 59 tasks (590 episodes): the excluded task's program calls `getattr`, which the sandbox refuses. J = judge, Esc. = escalations.

Arm	Succ	Rate	Steps/ep	Steps/succ	Wall/ep	Wall/succ	J s/ep	J calls/ep	Esc./ep
GPT-6 zero-shot, sample 1	116/600	0.193	831	4299	24.5	127	0	0	0
GPT-6 zero-shot, sample 2	91/600	0.152	806	5315	73.0	481	0	0	0
+ REA-V (both samples)	**158/600**	**0.263**	1586	6023	83.5	317	4.3	2.45	1.45
ASPIRE Fix-loop skills	391/590	0.663	1032	1557	27.1	40.9	0	0	0
ASPIRE Evo (11 tasks)	57/110	0.518	1310	2529	40.6	78.4	0	0	0
+ REA-M	424/590	0.719	1054	**1467**	41.6	57.9	0	0	0.01
+ REA-V	**442/590**	**0.749**	1883	2513	90.4	120.7	3.9	2.10	1.10

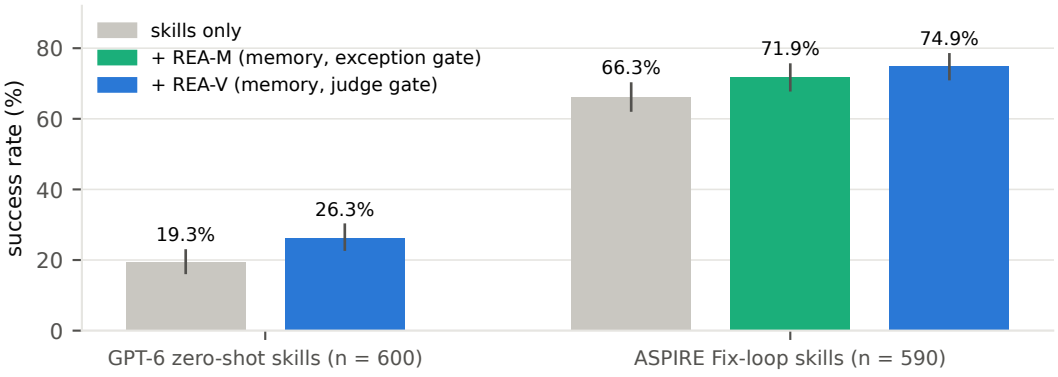


Figure 10: REA improves skill libraries of very different quality (Table 12; bars are Wilson 95% intervals).

J BEHAVIOR-1K AND ROBOTWIN 2.0

On BEHAVIOR-1K the three-level contract, the episode runner and the final score (ASPIRE's own `task_completed()`) are unchanged from LIBERO; only the backend differs. Tasks: `trash` (pick up the soda can) and `radio` (pick up the radio), R1Pro, 2025-challenge instances, Isaac Sim 4.5 and OmniGibson 3.7.2.

Held-out instances 1–10 (Table 14, top). One RTX PRO 5000 host, a 1,500 s budget per episode enforced at the API boundary, route memory and judge threshold from development instances 26–35. The REA arm routes over two frozen programs, a GPT-6 program (zero-shot sample or ASPIRE-style

Fix round, selected on development) and a program written offline by the OpenAI Agents SDK agent, with an agent choosing each dispatch (`jev-1.13.0`), the visual judge gate (`deepseek-flash`) and the ladder, so the full router is exercised. The Agents SDK agent runs `deepseek-v4-pro`, because the model relay serves GPT models only to the Codex client; Codex + GPT-6 is the Codex CLI harness with `gpt-6-astra` at high reasoning effort.

RoboTwin 2.0 (Figure 6). Every method runs the same 100 frozen bimanual scenes in SAPIEN (50 tasks, one clean and one randomized scene each, identical seeds and instructions) with `gpt-6-astra` at medium reasoning, at most 2,048 output tokens per request and at most 40 action decisions per episode. With no skill library, a fast reflex model is REA's System 0 skill and the router hands single decisions to a deliberate model. ASPIRE is a same-budget transfer (code execution with in-episode helper functions, no development-set training), not its full training protocol. REA and ASPIRE ran all 100 scenes in one environment; the Agents SDK arm comes from an earlier round. Successes (clean / randomized): REA 27 / 28, ASPIRE 25 / 26, Agents SDK 22 / 24. Paired, REA alone solves 10 scenes and ASPIRE alone 6 (p=0.45); against Agents SDK the split is 14:5 (p=0.06).

Development instances 26–35 (Table 14, bottom). One H20 GPU and 40 cores per lane, a 2,000 s cap per skill run and a 3,600 s episode budget. Each episode replays one frozen skill written by ASPIRE's interactive agent on these instances (K=0), the one route memory ranks first among three candidates on the same instances; these are development-stage latency numbers, and the agent's time to author the replayed skills is not counted.

Table 14: BEHAVIOR-1K. Top: held-out instances 1–10 (successes of 10; program wall time per episode, trash / radio). Bottom: development instances 26–35; elapsed time includes reset and scene loading, and for the interactive agent it is the elapsed time per completed trial, including Isaac start-up and timed-out attempts. The development frozen skill is the best of three candidates on these instances (they average 4.0/10 on trash and 3.3/10 on radio), so its success is optimistic; per attempt, the interactive agent takes 2.1× (trash) and 3.5× (radio) the replay's elapsed time. Best per column in bold.

Held-out arm (instances 1–10)	trash	radio	Total	Wall/ep (s)
Codex + GPT-6 (Codex CLI harness)	0/10	4/10	4/20	**471 / 310**
ASPIRE human-written programs	7/10	3/10	10/20	548 / 868
OpenAI Agents SDK agent, online	6/10	6/10	12/20	866 / 1107
REA (agent dispatch + judge gate)	**7/10**	**7/10**	**14/20**	787 / 966

Development arm (instances 26–35)	Task	Succ	Steps/succ	Wall/succ (s)	Elapsed/succ (s)
Human-written reference program	trash	5/10	7795	**1445**	**1565**
Interactive agent (agent in the loop)	trash	4/10	–	–	6438
REA, frozen skill (K=0)	trash	7/10	**6887**	1611	1734
Human-written reference program	radio	2/10	27,947	5547	5848
Interactive agent (agent in the loop)	radio	**4/10**	–	–	7281
REA, frozen skill (K=0)	radio	**4/10**	**9193**	**1945**	**2097**

K Same-Pool Baselines and Development Cost

Table 1 compares REA with each skill author in its pool run alone on the same 3,000 episodes. The *random author per task* row is the expected success of choosing one of the three authors uniformly per task; the *hindsight oracle* picks, for each task, the author with the most held-out successes (a ceiling that no selector fitted on development data can reach in expectation). *Route memory only* counts the 16 escalated episodes of REA-M as failures: their first run raised an exception, and 5 of them succeeded only after escalation, so K=0 reaches at least 2,871 of 3,000. *Aligned tasks*: on 37 tasks the pool holds exactly the programs the single-author rows ran, byte-identical for Codex and Agents SDK and the same program files for ASPIRE Fix (one task has no ASPIRE program in either); there the oracle is over the whole pool. On the other 23 tasks the pool's Codex slot holds a

harness-generated program (14 tasks) or its ASPIRE slot an ASPIRE Evo winner (10 tasks; one task has both), programs no single-author row ran. Table 15 compares the development budgets.

Table 15: Development budgets on LIBERO-Pro (60 tasks). Authoring episodes are the development evaluations the coding agents ran while writing their programs; route-memory episodes replay frozen programs on seeds 61–70.

Component	Dev. episodes	Input tokens	Dev. success of result
Codex + GPT-6 authoring	1,328	89.0M	75.0% (600 ep.)
Agents SDK + GPT-6 authoring	1,249	28.8M	95.3% (600 ep.)
ASPIRE Fix loop and Evo search	not recorded	not recorded	72.0% (590 ep.)
REA route memory (all 179 programs)	1,790	0	–

L Statistical Analysis

Every comparison is paired on identical (task, seed) episodes. Table 16 lists the paired success contrasts reported in the main text: exact two-sided McNemar tests, paired 95% intervals for the difference in success (Wald on the discordant pairs), and Holm-adjusted p-values within each question's family of tests. Every headline contrast survives the adjustment; the matched-subset gain over the best single skill does not (adjusted p=0.12), and the text reports it as unadjusted. Control-time intervals in Section 4.3 resample the 60 tasks. On one library and router, success does not depend on the host: seeds 1–10 give 550/600 on the H20 host and 546/600 on the PRO 5000 host (6:10, p=0.45).

Table 16: Paired success contrasts. Δ and the interval are in percentage points; b:c = episodes solved only by the first arm : only by the second.

Q	Contrast	n	b:c	Δ	Paired 95% CI	p	Holm p
Q1	REA-M vs Agents SDK (full set)	3000	86:51	$+1.2$	$[+0.4, +1.9]$	0.004	0.004
Q1	REA-M vs Agents SDK (aligned 37 tasks)	1850	54:25	$+1.6$	$[+0.6, +2.5]$	0.001	0.003
Q1	REA-M vs Codex (full set)	3000	669:25	$+21.5$	$[+19.9, +23.0]$	$< 10^{-4}$	$< 10^{-4}$
Q2	REA-M vs blocking advisor	3000	90:37	$+1.8$	$[+1.0, +2.5]$	$< 10^{-4}$	$< 10^{-4}$
Q2	REA-A vs REA-M	3000	26:26	$+0.0$	$[-0.5, +0.5]$	1.00	1.00
Q3	REA-V vs ASPIRE Fix skills	590	73:22	$+8.6$	$[+5.5, +11.8]$	$< 10^{-4}$	$< 10^{-4}$
Q3	REA-M vs ASPIRE Fix skills	590	49:16	$+5.6$	$[+3.0, +8.2]$	$< 10^{-4}$	0.0001
Q3	REA-V vs REA-M (ASPIRE skills)	590	37:19	$+3.1$	$[+0.6, +5.5]$	0.022	0.022
Q3	REA-V vs zero-shot skills	600	50:8	$+7.0$	$[+4.6, +9.4]$	$< 10^{-4}$	$< 10^{-4}$
Q4	REA-V vs best single skill (matched)	72	10:2	$+11.1$	$[+2.0, +20.2]$	0.039	0.12
Q4	REA-V vs REA-M (matched)	72	12:1	$+15.3$	$[+6.1, +24.4]$	0.003	0.014
Q4	REA-V vs REA-V w/o memory (matched)	72	15:6	$+12.5$	$[+0.4, +24.6]$	0.078	0.16
Q4	Blind retry vs its first skill (matched)	72	12:9	$+4.2$	$[-8.3, +16.6]$	0.66	0.66

M SUCCESS, COST AND LATENCY

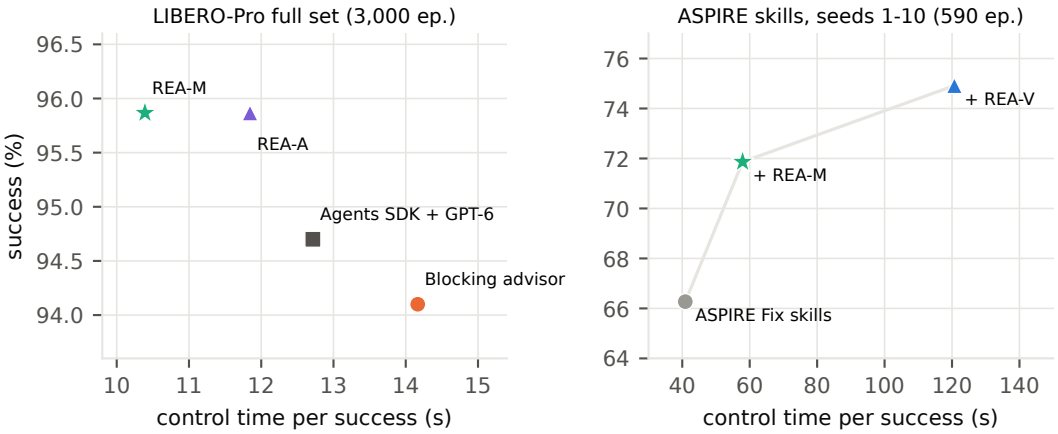


Figure 11: Success against control time per success. Left: LIBERO-Pro full set (one host; up and to the left is better); REA-M dominates every other arm. Right: ASPIRE skills on seeds 1–10; REA-M and REA-V extend the Pareto front of the single skill, REA-V buying $+3.1$ pp for $2.1\times$ the wall time per success.

N MODELS AND CONFIGURATION

Table 17: Models, versions and budgets. Every run plan records these values and the hashes of its frozen inputs.

Component	Setting
Skill authors	ASPIRE Fix loop / Evo search: Codex `gpt-5.6-sol`, high effort; Codex CLI and Agents SDK pipelines: `gpt-6-astra`, high effort; zero-shot: `gpt-6-astra`, two samples per task
Perception and motion	SAM 3, ContactGraspNet, PyRoki IK, blocking joint-position primitives; sandbox rejects identifiers that reach simulator state (e.g. `getattr`)
Visual judge	`deepseek-v4-flash-vision-exp` (matched subset), `deepseek-flash` (LIBERO-Pro): default prompt, episode-initial frame, 800×512 agentview, threshold 0.6, 200 max tokens; frozen on 216 development labels
Dispatch agent (REA-A)	`jev-1.13.0`, text only (program sources, development statistics, proprioception, exception history)
Blocking advisor (Q2)	`deepseek-flash`, text only, every 8 boundaries of 100 control steps, 30 s deadline
Caption bridge (Figure 5)	`qwen3-vl-flash` scene descriptions
Budgets (LIBERO)	4,000 control steps, 900 s per episode, at most three skill runs ($K=2$; $K=1$ for the exception-gated arms of the matched subset and the skill-author study)
Budgets (BEHAVIOR)	held-out: 1,500 s per episode; development: 2,000 s per run, 3,600 s per episode
Hosts	H20 (matched subset, skill-author study, Codex, ASPIRE and legacy full-set arms, BEHAVIOR development); RTX PRO 5000 (SDK full-set arms, speed study, placement ablation, BEHAVIOR held-out)

O REAL-ROBOT TASK: WALK TO AND PUSH A CHAIR (UNITREE G1)

A Unitree G1 humanoid, secured by a gantry, starts in front of an office chair and receives the instruction *take two small steps forward, push the chair, then retract the arms and stand still*. Every method uses the same robot, the same model (`deepseek-flash`, low reasoning) and the same four System 0 skills: walk forward (0.25 m/s for 1 s), the robot's native two-arm forward push, arm release and stop. ASPIRE writes one program with a single model request (code as policy); Codex + `deepseek-flash`, run as a separate campaign, writes a program with the Codex CLI; the Agents SDK agent calls the skills as tools (2–3 model requests); REA dispatches each skill at a skill boundary from the head and wrist camera images (4 decisions). Chair contact and displacement are not measured automatically, so, as Section 3 prescribes for real robots, SCORE is the operator's judgement on site, recorded after each trial and kept separate from the automatic logs. Three ASPIRE

failures and one Agents SDK failure stopped at a bridge error and were recovered to a safe stand; REA's first trial resumed from its completed walks after a failed status query.

Table 18: Unitree G1 walk-and-push-chair task, operator-judged success. Fisher exact tests against REA; one REA trial in Figure 12.

Method	Successful trials	Success (%)	p vs. REA
ASPIRE (code as policy)	2/10	20	0.023
Codex + `deepseek-flash` (Codex CLI)	4/10	40	0.17
OpenAI Agents SDK (tool agent)	5/10	50	0.35
REA	**8/10**	**80**	–



Figure 12: REA on the Unitree G1 walk-and-push-chair task (one successful trial, external camera). (a) Action sequence with elapsed time. (b) Initial, during-push and final frames: the chair moves toward the camera.

P Real-Robot Latency Check (Unitree G1)

Our Unitree G1 service, which does not use the router, computes brain decisions ahead of the request: a one-token vision decision (`deepseek-v4-flash-vision-exp`) as the brain, an in-process fused safety gate (robot state, RealSense depth and Livox lidar vetoes) and locomotion/arm actions as the executor. In *pipeline* mode a decision loop runs ahead of requests and the service returns the newest decision, 0.04–0.79 s old, which the gate re-checks against a live depth capture at execution (15–40 ms, not part of the request latency). Table 19 reports medians over 5–8 trials on the robot's onboard computer (Jetson) with live sensors. This measures latency only, not task success.

Table 19: Request latency on a Unitree G1 (medians over 5–8 trials). The two right columns are the same service; only pipelining differs. The pipeline total is request latency; the motion RPC (7–45 ms) and the live depth re-read (15–40 ms) follow at execution.

Stage	Original bridge	Fast harness	Service, single-shot	Service, pipeline
Capture	1.1–1.7 s	18 ms	14–36 ms	off the request path
Model	1.8–2.4 s	1.1–1.5 s	0.80–1.25 s	off the request path
Fused gate	2.3 s	4–10 ms	0.6–2.2 ms	0.6–2.2 ms
Motion accepted	1.5 s	7–45 ms	7–45 ms	7–45 ms (at execution)
Request latency	≈5 s	1.17–1.9 s	0.95 s	**1.0–3.6 ms**

Q An Agent as Its Own Cerebellum: the One-Token VLM Baseline

To test whether System 2 can also serve as System 0, we measured a direct vision-language policy on LIBERO: the same VLM used as our visual judge (`deepseek-v4-flash-vision-exp`) receives a 256-pixel RGB montage (external and wrist camera), the task instruction and proprioception, and emits exactly one action token per decision from a 15-token Cartesian/gripper vocabulary; each token is applied for five 20 Hz control steps (Table 20).

Table 20: One-token VLM baseline (official held-out initial states; this policy uses its own 600-step horizon, not the 4,000-step REA protocol).

Condition	Episodes completed	Successes	Rate
Pro (6 suites, 600-step horizon)	1,987	1	0.0005

Diagnosis. The two lateral tokens accounted for 89% of all outputs, the descend token was almost never chosen, and most episodes never descended at all; a replay of one episode's token sequence shows the end effector never came closer than 25 cm to the target object. The single success is a lateral sweep that happened to flip a stove knob. The cause is grounding, not actuation: the prompt described actions in world axes that the model cannot map onto either camera view (on the external view world $+x$ is screen-down at about 1 pixel per centimetre; on the wrist view left and right are mirrored), and the model fell back on a keyboard-style prior for the token letters. A scripted controller with privileged object poses but the *identical* token semantics, step size and horizon completed the pick-and-place tasks on 7 of 7 seeds within 36–63 tokens, so the action interface itself is sufficient. The placement ablation (Figure 5) adds a second, richer interface (3 cm / 15° Cartesian IK steps, a gripper command, scene captions, up to 80 decisions): both the agent and a text model solve 0 of 600 LIBERO-Pro episodes with it. This motivates the division of labour in REA: the VLM is used where it is reliable (a binary before/after judgement at a skill boundary) and continuous control is left to System 0 skills.