Where the Harness Meets the Physical World: A Survey of the Physical Agentic Harness

Physical Intelligence Team, Shanghai AI Laboratory

Abstract

Embodied foundation models now allow robots to act on open-ended instructions, yet success on long-horizon physical tasks also depends on the runtime around the model, which organizes the task, invokes skills, guards execution, and retains experience. Digital agent research treats this runtime, known as the harness, as a first-class design object that can be engineered and even improved from experience, since the same model can perform very differently under different harnesses. Such harness designs do not carry over directly, because physical execution is embodiment-specific, partially observed, time-constrained, and often irreversible, which also makes experience costly to learn from. Embodied AI systems increasingly build similar runtimes, yet the field still lacks a unified view of them. This survey treats the physical agentic harness as a research object in its own right. We define it as the runtime system that maintains task and execution state for an embodied foundation model and mediates the model's interaction with the physical world through one or more physical bodies. We then review over 200 works and organize them by six functional responsibilities: task orchestration, digital resources, embodied capabilities, execution governance, digital-physical interaction, and persistent evolution. This organization offers a common lens for comparing methods across communities, making explicit how model decisions reach physical execution and how grounded evidence flows back. The reviewed benchmarks and applications provide limited evidence of individual runtime contributions, particularly the effects of execution governance and sustained gains from persistent evolution. We close with open problems in model-harness co-design, capability composition, real-time decision making, distributed execution, and continual improvement from physical experience.

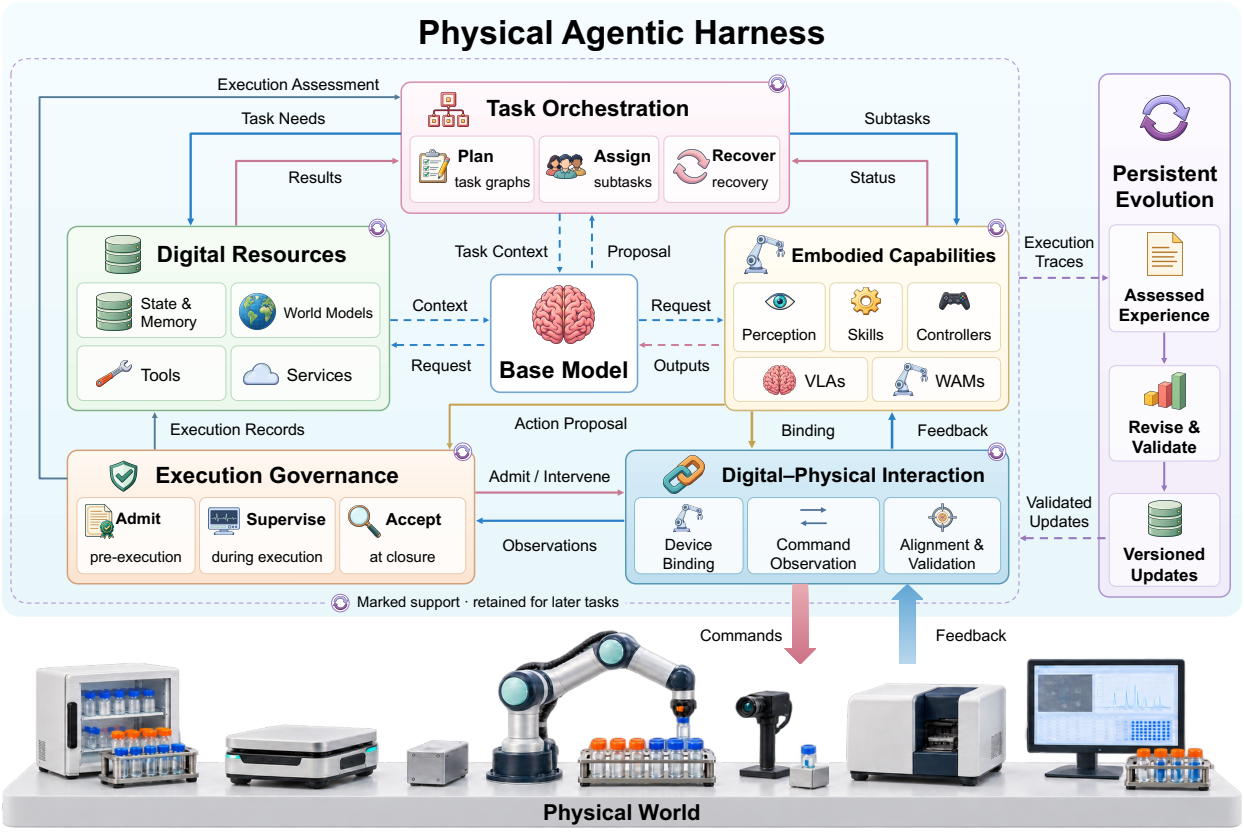


Figure 1 Functional overview of the Physical Agentic Harness. The harness links an embodied foundation model to the physical world through five interacting responsibilities within a task. Persistent evolution uses assessed execution experience to revise runtime support retained for later episodes.

Contents

1 Introduction . . . 3

2 Definition and Scope of the Physical Agentic Harness . . . 4
- 2.1 Physical Requirements of Runtime Coordination . . . 5
- 2.2 Definition and Runtime Boundary . . . 6
- 2.3 Functional Responsibilities of the Physical Agentic Harness . . . 7

3 Task Orchestration in a Physical Agentic Harness . . . 8
- 3.1 Task Structuring . . . 8
- 3.2 Subtask Management . . . 9
- 3.3 Task Recovery . . . 12

4 Digital Resources in a Physical Agentic Harness . . . 13
- 4.1 Digital Resources for Knowing the Situation . . . 13
- 4.2 Digital Resources for Predicting Outcomes . . . 15
- 4.3 Digital Resources for Computing Solutions . . . 16
- 4.4 Using Digital Resources: Selection, Execution, and Feedback . . . 17

5 Embodied Capabilities in a Physical Agentic Harness . . . 18
- 5.1 Perception and Action Interfaces . . . 19
- 5.2 Action Capability Implementations . . . 20
- 5.3 Capability Management . . . 22
- 5.4 Capability Composition and Adaptation . . . 23

6 Execution Governance in a Physical Agentic Harness . . . 24
- 6.1 Governance State and Executable Constraints . . . 24
- 6.2 Before Execution: Action Admission . . . 25
- 6.3 During Execution: Supervision and Intervention . . . 26
- 6.4 Closing Execution: Outcome Acceptance and Audit . . . 27

7 Digital–Physical Interaction in a Physical Agentic Harness . . . 28
- 7.1 Resolving Operations to Physical Interfaces . . . 28
- 7.2 Dispatching Operations and Interpreting Evidence . . . 30
- 7.3 Maintaining Spatial and Temporal Reference . . . 30
- 7.4 Recovering from Interface Faults . . . 31

8 Persistent Evolution in a Physical Agentic Harness . . . 32
- 8.1 Targets of Persistent Evolution . . . 32
- 8.2 Evolution Process . . . 32
- 8.3 Evolution Methods . . . 34
- 8.4 Self-Evolution . . . 37

9 Evaluation and Applications of Physical Agentic Harnesses . . . 38
- 9.1 Task and Capability Benchmarks . . . 38
- 9.2 Benchmarks for Coordination, Recovery, and Governance . . . 39
- 9.3 Applications in Physical Environments . . . 43

10 Open Problems and Research Directions . . . 46

11 Conclusion . . . 48

Author List . . . 48

1 Introduction

Embodied foundation models have substantially expanded what robots can do from open-ended instructions. They now span vision-language-action models (VLAs) [1, 2, 3, 4], embodied reasoning models [4, 5, 6, 7], world models [8, 9], and world action models (WAMs) [10, 11, 12], and robot systems increasingly combine several of them [5, 13]. Whichever model is used, each call returns an output such as actions, a plan, or a predicted future, based on the context available at that moment. On short, well-specified tasks this output can be close to the whole task, whereas on long-horizon physical tasks it covers only one decision within a longer process.

Carrying a task through that process requires sustained interaction with the physical world. After each operation, the robot has to check what actually changed, keep its task state consistent with the scene, and decide whether to continue, retry, or replan [14, 15]; when several robots share a task, the same state has to stay consistent across bodies. Systems that add closed-loop feedback, failure explanation, or failure detection around existing models report clear gains in task success [14, 16, 17]. Long-horizon reliability depends on both model capabilities and the organization of this interaction.

In digital-agent research, this surrounding runtime is increasingly described as the *harness* [18, 19, 20]. It maintains context and execution state, exposes tools, and manages calls and their results. The model supplies learned task capabilities; the harness organizes their use under current execution conditions. This boundary is functional: runtime decisions can themselves use learned models, and the division can change with the model and its interfaces. Changing the harness alone can improve the performance of a fixed model [18, 21]. Self-Harness illustrates this complementarity: the fixed base model proposes harness revisions, which undergo regression testing before adoption [22].

Physical execution places additional demands on this model–harness relationship. In isolated software environments, candidate harnesses can be compared through repeated, resettable trials [20]. A robot acts through body-specific interfaces and relies on partial observations. The scene may change during inference, so the runtime must relate proposed actions to evidence that remains valid when they execute [15, 23]. Failed actions may also leave effects that cannot be undone; retrying then requires reassessing the resulting physical state [14].

The cost of physical experience limits how broadly deployment failures can be covered through training [24, 25]. Runtime checks and recovery offer a way to address some failures without retraining the base policy, provided the deviation is detectable and an appropriate capability is available. Harness VLA illustrates this approach by coordinating a frozen VLA with memory and analytic primitives [26]. Missing physical capabilities may still require learning, and runtime revisions also need validation. The two approaches can reinforce each other: Guava uses a shared harness to collect interaction data and train a compact model [27]. Comparisons therefore need to account for the physical trials required to develop and validate each change.

Robotics has addressed parts of this problem for decades, under different names and in different communities. Three-layer architectures separate deliberation, sequencing, and reactive control [28], skill frameworks make preconditions and required resources explicit [29], and runtime-assurance systems monitor physical state and switch to safe controllers when needed [30]. Foundation-model systems extend these ideas with tool use, programmatic skills, and feedback-based replanning [31, 32, 14]. More recently, several embodied systems have begun to describe their runtime as a harness or a robot operating layer, using it to manage bodies and skills, close inspection loops, or revise its own support from experience [33, 27, 26, 34, 35, 36, 37], and Lee et al. [38] describe robot middleware as a harness for Physical AI. These lines of work tackle overlapping problems, yet they use different vocabularies, draw the system boundary in different places, and are rarely compared with one another. Comparing these mechanisms requires a definition that separates runtime support from model capabilities and makes the physical conditions of execution explicit.

Harness-oriented embodied systems are appearing quickly, and recent studies increasingly compare runtime designs around fixed models [39, 40, 41]. Their protocols differ in capability libraries, privileged information, retry budgets, and adaptation costs, so the resulting gains require a common basis for interpretation. Existing reviews cover neighboring ground, including robot learning and embodiment adaptation [42, 43], world action models [44], externalized resources in digital agents [45], and harnesses for large language model (LLM)

agents [46]. Building on these perspectives, we use runtime responsibility as the unit of comparison. This connects mechanisms from robot architectures, skill frameworks, runtime assurance, and foundation-model agents through the physical execution problems they address. Our focus is how task state, capability interfaces, and outcome checks work together under embodiment and timing constraints, and how revisions to this support are evaluated across episodes.

We define the *Physical Agentic Harness* as the runtime system that maintains task and execution state for an embodied foundation model and mediates the model's interaction with the physical world through one or more physical bodies. The definition applies across software architectures and organizes the literature by six functional responsibilities: task orchestration, digital resources, embodied capabilities, execution governance, digital–physical interaction, and persistent evolution.

These responsibilities connect execution within a task to revision across episodes. Task orchestration uses model proposals and observed outcomes to manage task progress. It draws on digital resources for information and computation, and on embodied capabilities for perception and action. Execution governance constrains the physical use of those capabilities and assesses their outcomes. Digital–physical interaction carries authorized operations to devices and returns evidence for governance and task reassessment. These exchanges form a task-level feedback loop whose implementation can vary across systems. Persistent evolution uses assessed experience from this loop to revise support retained for later episodes. It can affect any of the other five responsibilities and may include human-guided updates. Figure 1 summarizes these relationships around the embodied foundation model.

We review over 200 works, using benchmarks and applications to assess evidence for the six responsibilities. Figure 2 provides an overview of representative systems and works across these responsibilities, alongside the benchmarks and applications covered in the survey. The groupings are non-exclusive, and including a work does not imply that it implements a complete harness.

This organization supports two linked comparisons: how runtime mechanisms connect model decisions to physical execution, and what the reported experiments establish about those mechanisms. We examine the information, capabilities, and execution conditions changed by each comparison, so that a whole-system gain is distinguished from evidence for an individual runtime contribution. Our review of the evaluation literature shows that harness comparisons need to examine runtime behavior alongside task outcomes [47], and that recent fixed-model and governance comparisons establish benefits within specific protocols [39, 40, 41]. The remaining gaps concern attribution across interacting components, generalization beyond development conditions, and sustained physical improvement under independent outcome assessment [48, 49]. These gaps frame the open problems we discuss: model–harness co-design, capability composition, real-time decision making, distributed execution, and continual improvement from physical experience.

The rest of this survey is organized as follows. Section 2 defines the Physical Agentic Harness and draws its runtime boundary. Sections 3–8 then follow the six responsibilities in Figure 1, from task orchestration to persistent evolution, and Section 9 relates them to evidence from benchmarks and applications. Section 10 discusses open problems and Section 11 concludes.

2 Definition and Scope of the Physical Agentic Harness

Physical execution creates requirements for coordination that help define the Harness boundary. The six responsibilities in Figure 1 organize these requirements; Figure 2 maps the reviewed literature to them.

Review Methods

Literature Scope We include work that explains a mechanism relevant to at least one Harness responsibility or supplies evidence for its evaluation. Classical robotics work provides technical foundations; its inclusion does not imply that it is itself a foundation-model Harness. Studies confined to digital environments are used as methodological comparisons and are identified as such. The coverage is not an exhaustive systematic review. A targeted full-text update checked on 9 November 2026 covers 36 priority studies, including earlier omissions and recent preprints. The update date identifies when this selected corpus was checked; it does



Figure 2 Overview of representative systems, frameworks, and research organized around the six functional responsibilities of the Physical Agentic Harness, together with representative benchmarks and applications. Individual works may span multiple responsibilities; their inclusion does not imply implementation of the complete harness.

not establish an exhaustive search cutoff for the earlier manuscript. Inclusion requires a relevant runtime mechanism, a controlled comparison, or an evaluation protocol; training-centric and digital-only studies are used only for the boundary or methodological comparison they support. For experimental claims, we distinguish fixed-model comparisons from changed controllers or training, simulation from hardware, and development feedback from independent evaluation. Preprints are cited as such; retention, task progress, and cumulative recovery are not treated as interchangeable with held-out task success.

2.1 Physical Requirements of Runtime Coordination

Physical execution requires the runtime to connect each proposed operation to the conditions under which it can proceed. The first question is whether a capability can be used with the available robot and workspace. A manipulation skill may depend on a particular sensor configuration or control interface. Its description must expose these requirements so that the runtime can assess a concrete request [31, 50].

The assessment also depends on the context of each observation. An object pose must be interpreted through a coordinate transform related to the robot [51, 52]. Its acquisition time matters because the object may move while the model is computing [15]. The runtime should retain frame and timestamp metadata and obtain an updated observation or another valid state estimate when existing evidence is incomplete or stale.

The runtime must relate each operation to the dependencies of the wider task. Skill selection and feedback-based planning distinguish a proposed action from evidence that its required effect occurred [31, 53]. As an illustrative example, consider a robot transferring a prepared sample to an instrument. Measurement can begin only after the sample reaches the required position; issuing the transfer command does not establish that condition. More generally, multi-robot execution must account for shared resources and dependencies between operations [54].

Task readiness and execution permission require separate decisions. An operation may fit the workflow and remain physically feasible while violating a safety constraint. Runtime checks must therefore determine whether it may start under the applicable limits [30]. Those limits can remain relevant after execution begins, so new evidence may require supervision or intervention [15].

The runtime must also decide how to continue after an operation ends. A failed or interrupted operation may already have changed the sample or its surroundings. Recovery must use the state left by that attempt instead of assuming that the action had no effect [14, 53]. Retaining outcome information allows the runtime to identify which parts of the task remain valid and which require reassessment.

These requirements determine the state and coordination functions in the definition below.

2.2 Definition and Runtime Boundary

> **Definition.** A *Physical Agentic Harness* is the runtime system that maintains task and execution state for an embodied foundation model. It coordinates the resources and capabilities used during a physical task and mediates the model's interaction with the physical world through one or more robot bodies.

The runtime can also interact directly with task-relevant instruments and environmental devices through managed physical interfaces.

Here, an embodied foundation model is a foundation model used for reasoning, perception, or action in a physical agent. We use *Base Model* for the model whose task-directed behavior the Harness supports. It can be a language or vision–language model invoking robot capabilities, or a model that generates robot actions. A separate planning model and action model are therefore not required. Other models can also be managed as resources or capability implementations. For each comparison, the Base Model and any separately managed models must be identified consistently, so that a change in system description does not reclassify the same weight update as a Harness-only change.

State management provides continuity across model invocations and capability calls. Task state describes the workflow and its progress, including responsibility for individual subtasks. Execution state links operations to their reported status and observed outcomes. Together, these records allow later decisions to account for earlier execution. The distinction describes how runtime information is used; it does not require separate data structures.

Mediation connects a model proposal to the support needed for physical execution. Digital resources provide information and computation. Embodied capabilities expose callable perception and action implementations. The Harness coordinates their use for the current task and applies the relevant execution constraints. Observations then update the context used by the model and by other runtime components.

A component contributes a Harness function through task-state management, coordination of resources or capabilities, or mediation of physical execution. For a managed capability, this support can include its callable interface, implementation selection, invocation state, and execution checks. One component can implement both a learned mapping or control law and this surrounding support; the mapping or control law supplies the capability being managed. Its location in the robot software stack does not determine which role it performs.

Harness-managed state can persist beyond the current episode. An episode begins when a task goal is accepted. It ends when the final outcome for that goal is accepted or the task is terminated. Accepting an individual operation or subtask does not by itself end the episode. Updates used only within this interval support the current episode. Persistent Evolution concerns revisions to runtime support retained for later episodes, not ordinary log storage. These episodes may repeat the same task or involve different tasks; persistence across episodes does not by itself establish generalization to new task types. Retention establishes availability, adoption establishes later use, and improvement requires evidence of benefit; a revision can be studied without already demonstrating improvement. Section 8 examines these distinctions.

2.3 Functional Responsibilities of the Physical Agentic Harness

The first five responsibilities distinguish the organization of a task, its information and computation, its callable behaviors, its execution constraints, and its device exchanges. Persistent Evolution examines how this support changes across episodes. We use these categories as one analytical organization of the literature. They allow implementations to combine responsibilities or distribute a responsibility across services, with execution order determined by task dependencies. The exchanges in Figure 1 illustrate these relationships without requiring every interaction to pass through the Base Model.

Task Orchestration maintains the workflow that relates operations to the task goal. It establishes subtask dependencies and conditions for progress, assigns responsibility, and uses observed outcomes to coordinate execution. When a required condition no longer holds, it selects a continuation that preserves valid progress and revises the workflow if needed. The resulting task requirements guide the use of Digital Resources and Embodied Capabilities.

Digital Resources supply information and computation for these task requirements. The Harness may retrieve maintained information, invoke a tool, or use a predictive model or simulator to examine a candidate solution. It must select resources that apply to the current task and provide the computing capacity needed to use them. Results should be retained with the context needed for later interpretation.

Embodied Capabilities expose callable perception and action implementations for physical execution. Their descriptions identify the sensing, control, and operating conditions required by an implementation. Capability management matches these requirements to the current request and prepares the selected implementation for use. Composition and adaptation can broaden its use while keeping the relevant conditions explicit. Physical actuation remains subject to Execution Governance.

Execution Governance determines whether and how a proposed physical operation may proceed. Admission grants permission under specified constraints. It does not settle whether the operation remains acceptable after execution begins. Supervision can therefore lead to intervention when new evidence warrants it. At closure, outcome acceptance assesses whether the observed effect satisfies the relevant criteria.

Digital–Physical Interaction carries authorized operations to the Physical World and returns observations and execution outcomes to the runtime. It maintains the device bindings, command and observation interpretations, and references needed to associate feedback with the operation that produced it. This responsibility connects the Harness to physical execution and supports the reassessment performed by Task Orchestration and Execution Governance.

Persistent Evolution uses assessed execution experience to revise components retained for later episodes. These revisions can affect the other five responsibilities, including settings used in Digital–Physical Interaction. Adoption occurs when a later episode uses the revision. Evaluation and version management relate later behavior to the revision that was actually adopted. Human participation may remain part of this process, while self-evolution concerns how much of the update process is automated.

These responsibilities interact through shared task context and execution evidence. Discovery, resource preparation, and device binding may precede action admission. Authorization constrains physical dispatch, and changes to the assessed configuration can require reassessment. The section order follows these functional relationships; actual execution can revisit them as conditions change.

3 Task Orchestration in a Physical Agentic Harness

Task Orchestration maintains the workflow and execution responsibility of a physical task. It records active subtasks, their assigned executors, and the conditions for starting and completing each stage. Observations and assessed outcomes update this state as execution proceeds. Changes in the body or environment can invalidate a dependency, require reassignment, or change the remaining procedure [55, 54, 56]. The task-level choice of what to attempt remains distinct from permission to execute an operation and acceptance of its observed outcome.

Physical changes determine how much of the task remains valid after an unexpected outcome. A completed operation may establish conditions for several later stages; a disturbance may affect one stage or a shared object used by multiple branches. Task Orchestration identifies these dependencies and preserves progress whose supporting conditions still hold [55]. Assignment and coordination then connect the revised task state to available executors, their timing, and shared resources [56, 54].

Task Structuring describes the stages and their dependencies. Subtask Management tracks progress, assignments, handoffs, and concurrent execution. Task Recovery seeks a feasible continuation after failure or interruption, revising the workflow when necessary. These are functional responsibilities within orchestration; an implementation may combine them in an executive, planner, or distributed runtime.

Figure 3 summarizes this view of task orchestration.

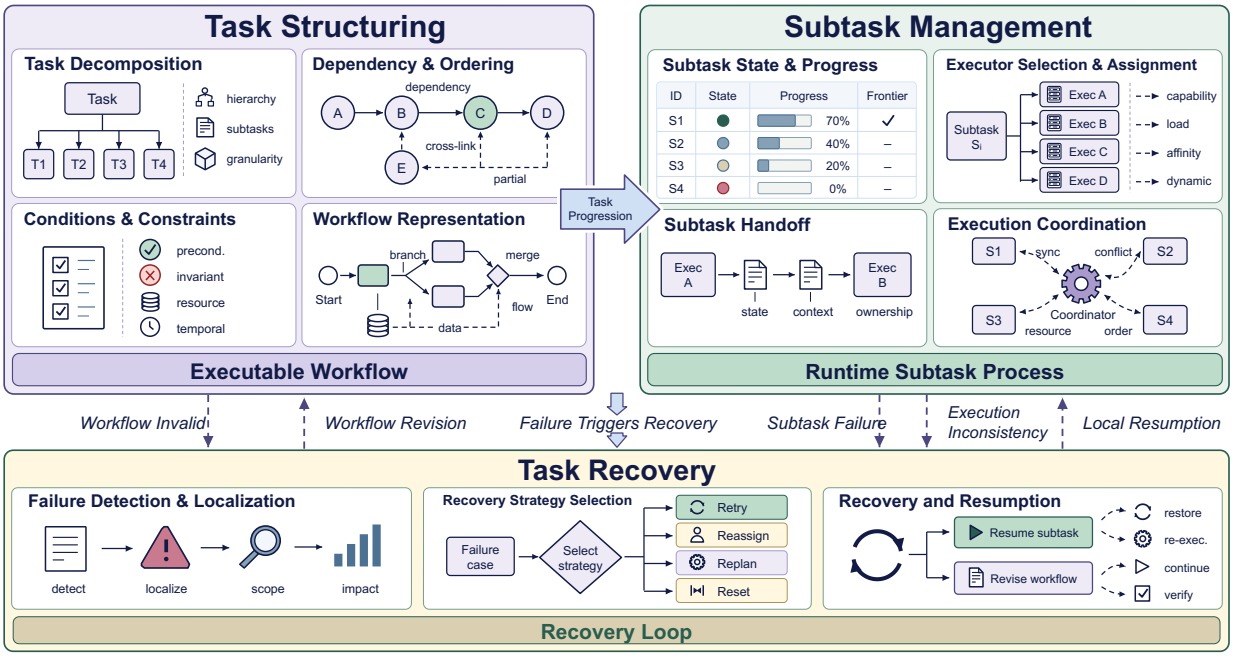


Figure 3 Overview of Task Orchestration in a Physical Agentic Harness. Task Structuring converts a task into an executable workflow, Subtask Management maintains progress and coordinates executors, and Task Recovery detects and localizes failures, selects a recovery strategy, and resumes or revises execution. Dashed feedback paths show how workflow invalidation, subtask failures, and execution inconsistencies return the task to a valid execution state.

3.1 Task Structuring

Task structure relates subtasks to the overall physical goal. It records their entry conditions, expected effects, dependencies, and completion criteria. A Base Model may propose this structure, while Task Orchestration retains it and uses observations to assess transitions. Subtask management then assesses whether current evidence supports the transition and assigns a ready stage to an executor.

Existing approaches represent such structure at different levels of formality. Language-based agents decompose instructions into semantically meaningful subgoals, whereas symbolic and neurosymbolic approaches expose predicates, action schemas, and state transitions. PDDL-ART constructs task-specific PDDL domains and problems from demonstrations and natural-language descriptions, then refines them through syntactic checks, semantic validation, and execution feedback [57]. Neurosymbolic embodied agents ground goal-relevant predicates into a symbolic initial state and use a PDDL transition model to constrain executable continuations [58]. These approaches illustrate that decomposition must be accompanied by conditions explaining when the resulting subtasks are applicable and what their execution should establish. Recent multi-agent systems such as MistyPilot make sensor-triggered skill binding and task-state management explicit in a physical social robot [59].

Dependencies also extend beyond sequential ordering. A subtask may require several predecessors, alternative branches may depend on observations acquired later, and some conditions must remain valid throughout an interval of execution. Zaman et al. [60] compile temporal specifications into automata and embed their guards into action schemas, allowing ordering, invariance, and liveness requirements to constrain task and motion planning. SUN Programs uses typed executable programs, predicates, and transition guards to encode task semantics [61]. Such representations make task-wide constraints explicit without requiring the high-level workflow to prescribe every local control decision.

Task structuring also maintains the relevance of this structure as execution unfolds. A changed observation need not trigger a new decomposition: it may only change a subtask's arguments or executor. Structural revision becomes necessary when evidence contradicts a dependency, invalidates a required subgoal, or changes the feasible continuation. ContextFlow illustrates this distinction through scoped updates that can retain a stage while changing its executor or preserve a validated prefix while repairing an unsupported suffix [55]. We use this distinction to separate ordinary subtask management from workflow-level repair, discussed further in Section 3.3.

TAMPAgent makes geometric uncertainty explicit in this process: the language model proposes a partial belief-space plan, while task-and-motion planning and perception tools bind constraints before execution [62]. Its feasibility checks are conditional on the planning model and tool estimates, rather than guarantees about unmodeled physical effects. LOGIV instead tracks causal dependencies between symbolic conditions, checking execution evidence and repairing the affected plan suffix while retaining still-valid progress [63]. These approaches place checks between a proposed symbolic step and its execution, with the result conditional on the modeled constraints and available observations.

The workflow may take the form of a symbolic plan, task graph, state machine, behavior tree, or executable program. For the Harness, the key property is whether it exposes the relationships required to inspect, advance, and revise the task. Task structuring provides the workflow structure maintained during the task; subtask management instantiates it through evidence-grounded state updates, execution assignments, handoffs, and coordination.

3.2 Subtask Management

Subtask management maintains the execution state, responsibility, and coordination of the subtasks within the overall workflow. Here, a subtask is a semantically meaningful stage with a local objective and conditions for entry, continuation, and completion. It may invoke one capability, compose several capabilities, or require multiple executors. The distinction from Section 3.1 is one of runtime responsibility: task structuring organizes the overall task, whereas subtask management determines what is currently supported, who should act, how responsibility is transferred, and how interdependent activities proceed.

Subtask State and Progress Subtask state records how a workflow stage is instantiated at a particular point in execution. It identifies whether the subtask is ready, active, completed, blocked, or no longer valid, together with its grounded arguments, assigned executor, and supporting observations. Together, these records define the current task frontier: established outcomes and supported continuations. Progress reflects assessed changes in task-relevant state. Elapsed time, issued commands, and local termination do not by themselves establish completion. Updated subtask state also supplies context for subsequent model proposals.

Maintaining this state requires evidence to be preserved and revised across subtask boundaries. Inner Monologue incorporates success detection and scene descriptions into planning feedback, allowing later decisions to reflect earlier physical outcomes [14]. ContextFlow relates the active stage to runtime evidence, remembered context, and the delegated executor [55]. Its task-frontier perspective makes disagreement among these sources an explicit orchestration problem. Complementing these mechanisms, Wang et al. [64] characterize long-horizon difficulty through the need to retain previously observed state, explore hidden state, and revise information that has become outdated, rather than through action-sequence length alone.

Execution status and semantic completion must remain distinct. PhyAgentOS evaluates execution sessions against evidence-grounded acceptance criteria and returns success, failure, or replan verdicts [56]. Here, an execution session records an attempt against a stated objective; the term does not identify a device connection. EmbodiedSkills similarly checks an active subgoal using fresh post-action observations before selecting the next semantic route [65]. At the orchestration level, these verdicts determine whether the subtask can be accepted and its successors reconsidered. An insufficiently supported outcome may instead require continued execution, additional observation, or recovery.

Partial observability further separates missing evidence from demonstrated failure. Tanaka et al. [66] maintain an evidence state containing exploratory subgoals, observed facts, and retained findings, and distinguish proceeding from further evidence acquisition or halting. Ojha et al. [67] treat newly discovered objects as replanning events, combining reconstructed visible state with completed-action history before revising the remaining sequence. Consequently, subtask progress can involve resolving uncertainty or revising stale assumptions without immediately advancing to another stage. Previously completed outcomes remain reusable only while the conditions needed by subsequent subtasks are still supported.

Executor Selection and Assignment Executor selection assigns a ready subtask to a robot, instrument, or other executor that can operate under the current conditions. Applicability checks establish whether a candidate can perform the operation; availability and existing commitments determine when it can take responsibility. The assignment records the objective, grounded context, expected outcome, and conditions under which responsibility begins and ends.

Classical multi-robot task allocation provides foundations for this decision. Gerkey et al. [68] distinguish allocation problems by the numbers of robots required per task, tasks supported per robot, and the temporal scope of assignment. Korsah et al. [69] extend the analysis to complex tasks with interdependent utilities, schedules, and constraints. In the present hierarchy, these allocation mechanisms apply to the subtasks produced by the overall workflow, including subtasks requiring a team rather than a single robot.

Selection depends on both subtask requirements and execution context. A candidate must have an applicable capability and be available when needed; location, workload, communication status, resource access, and existing commitments can also affect feasibility. Assigning an executor to one subtask may delay another or reserve an object needed elsewhere. These interactions couple executor assignment to scheduling.

Centralized approaches can select assignments and schedules over a shared state, whereas distributed approaches combine local evaluations through communication, auctions, or consensus. Choi et al. [70], for example, combine bidding and consensus to resolve conflicting assignments. Assignment can also remain provisional before execution begins. Garces et al. [71] consider scheduled and online requests together with compatibility, routing, service order, and time windows; their policy can wait or reconsider assignments that have not started. The methods differ in when an assignment becomes committed and which decisions remain revisable.

REA treats executor choice as a budgeted routing problem, using route memory and boundary judgments to decide whether to escalate between available executors [72]. Its 37-task aligned subset uses the same policy files as the baselines. In a separate judge-gating comparison, additional checking improves completion while increasing time per successful episode. DynaHarness separates slow semantic proposals from a faster deterministic controller with bounded execution leases [73]. Because the system relies heavily on analytic skills, attributing its gain to improved use of the frozen VLA requires accounting for their contribution.

Subtask Handoff Subtask handoff connects successive stages or transfers responsibility for an ongoing stage. A stage transition requires a physical state suitable for the next capability, even when the same robot performs both stages. A replacement executor needs the objective and completion criteria, together with grounded arguments and validated progress. Relevant observations and unresolved conditions inform its next decision. The assignment record identifies when responsibility changes. The handoff must also account for operations still active under the previous assignment.

A transfer within a subtask need not alter the high-level workflow. ContextFlow preserves a valid semantic stage while assigning it to an executor better matched to the current context [55]. This separates executor replacement from replanning: a new execution carrier can inherit the same objective and validated progress. Across successive subtasks, the central question instead concerns compatibility between the predecessor's outcome and the successor's entry conditions.

Local success does not guarantee this compatibility. Rui et al. [74] identify semantic handoff failures in which a skill satisfies its own termination condition but leaves the next skill unable to proceed. Navigation may, for example, make a target visible without placing it within manipulation reach. BATON addresses related transition problems through handoff-aware memory: handoff transitions restore entry conditions disturbed by a preceding subtask, while lookahead transitions choose strategies whose outcomes can be inherited by a successor [75]. Handoff can therefore require physical preparation as well as transfer of task information.

These conditions are checked before accepting the transition. The receiver's context should preserve the source and timing of observations, active constraints, and unresolved outcome assessments. If the physical starting state is unsuitable, the handoff can remain pending, request preparation, transfer responsibility elsewhere, or enter recovery. Stopping the previous executor alone does not establish readiness for the next stage.

Execution Coordination Execution coordination manages dependencies between concurrent operations. Robots and instruments may share objects, tools, or workspaces, so their timing must preserve the conditions required by each subtask. Observed progress determines when a dependent stage becomes ready and when a shared resource can be released.

Coordination can involve predecessor constraints, synchronization points, mutual exclusion, and shared-object handoffs. Physical Agentic AI expresses ordering, synchronization, resource dependencies, completion conditions, and execution invariants through workflow contracts [54]. CoCoBench distinguishes sequential ordering, mutual exclusion, and handoff coordination, and shows why aggregate task success can obscure duplicated work, ordering violations, resource contention, and unsynchronized transfers [47]. These relations require coordination across subtasks even when each executor performs its assigned operation correctly.

Asynchronous execution allows a subtask to proceed once its own dependencies are satisfied, without requiring every executor to reach a common global step. APEX-MR postprocesses multi-robot task and motion plans for asynchronous execution under uncertainty and evaluates the resulting coordination in cooperative assembly [76]. IndoorR2X dispatches dependency-ready task nodes to available robots in parallel and integrates completion events, collisions, and external-device updates through asynchronous monitoring [77]. These mechanisms link local completion events to the readiness of dependent subtasks while avoiding unnecessary global synchronization.

Shared resources, including passages and computational services, require ordered acquisition and release to prevent conflicting use and circular waiting. Chakravarty et al. [78] jointly formulate constrained resource assignment and continuous temporal ordering, making reservations part of the evolving schedule. When communication is intermittent, coordination may additionally rely on local commitments and later reconciliation. Cegarra et al. [79] use connectivity-aware graph extension and stable area partitions for decentralized exploration under communication and layout changes. Such mechanisms do not require every executor to possess a continuously complete global state.

The Zeno Research report on Zeno-1 [80] supplies a supplementary, non-peer-reviewed example. The report describes a shared policy operating independently on each robot using local observations and interaction memory. It illustrates coordination inside an embodied policy; it does not establish a general workflow-construction or task-allocation mechanism.

Coordination combines explicit workflow relations with local adaptation inside capabilities. The Harness manages the cross-subtask obligations left to it by each executor and reports invalid dependencies, resource conflicts, or failed handoffs to task recovery.

3.3 Task Recovery

Task recovery identifies a feasible continuation from the physical state left by execution. An unmet outcome, unavailable executor, or invalid handoff can affect one subtask or several dependent branches. Diagnosis identifies that scope so recovery can preserve progress whose supporting conditions still hold. Task Orchestration then chooses whether to retry, reassign, replan, suspend, or terminate.

Failure Detection and Localization Failure detection asks whether the conditions needed by the active stage or a successor remain supported. The completion and handoff checks above supply possible triggers. An unexpected observation can instead reveal missing evidence without invalidating the workflow, so diagnosis must distinguish an unsupported conclusion from a contradicted condition.

Diagnosis can require execution history rather than a single terminal observation. REFLECT converts multisensory robot experience into hierarchical summaries and uses progressive failure explanation to support corrective planning [16]. This provides a way to relate an observed failure to earlier actions or unsatisfied subgoals. For orchestration, such explanations are useful when they identify which task assumption should be reconsidered. The explanation remains a diagnosis whose causal claims require verification against execution evidence.

Monitoring can also identify violations during execution or anticipate impending ones. Code-as-Monitor formulates reactive and proactive failure detection as spatiotemporal constraint checking and uses VLM-generated code to evaluate those constraints [81]. The generated checks make the monitored condition inspectable. At task level, the detected condition identifies the subtask or dependency that needs repair.

Detection and diagnosis also have different timing requirements. RoboBRIDGE uses a lightweight success check during execution and invokes detailed diagnosis when a failure is detected with sufficient confidence [82]. This separates frequent monitoring from less frequent recovery reasoning. Retaining the detected condition and its context can reduce the need to repeat full task-level reasoning at every control step.

Localization then determines the smallest task region whose validity is no longer supported. ContextFlow frames this as misalignment among the active stage, evidence, memory, and delegated execution, and distinguishes a valid stage with an unsuitable executor from an unsupported workflow suffix [55]. Extending that distinction to a general task graph, we separate the failed action or executor from the subtasks that depend on its outcome. A disturbed shared object can invalidate several branches, whereas an unrelated completed branch may remain usable. Localization follows affected conditions and dependencies, preserving any later stages that remain valid. Its output is a repair scope, a record of still-valid progress, and the evidence needed to reassess the remaining task.

Recovery and Resumption Recovery selection should match this scope. Additional sensing is appropriate when the obstacle is uncertainty; retry or parameter adjustment may suffice when the subtask and its entry conditions remain valid. Reassignment can address an unsuitable executor without changing the objective. Broader disruption can require physical state restoration, local workflow repair, or replanning. The alternatives are selected according to the failure scope. For example, a known-invalid entry state requires repair before retrying.

RoboBRIDGE makes recovery scope explicit through four targets: retrying the current primitive, regenerating its trajectory while retaining the plan, replanning from current perception, and forcing renewed perception before replanning when scene state is unreliable [82]. These operations distinguish failures of local execution from failures of planning context. ContextFlow provides a complementary task-state view by preserving validated progress while refining a stage, transferring its executor, or repairing only the unsupported suffix [55].

Physical state restoration requires a different intervention from issuing the same action again. FLARE distinguishes Retry behavior for execution deviations from Reset skills for states that the ordinary task policy

cannot handle. It trains retry behavior with perturbation-and-bridging data and uses an online multimodal monitor to arbitrate between task execution and object-centric reset skills [83]. This runtime choice seeks a task-valid configuration from which execution can resume.

Recovery can also be prepared before failure occurs. AgentChord augments a nominal task graph with anticipated failure modes, executable triggers, recovery nodes, and connections back to feasible downstream nodes [84]. This connects recovery construction to the high-level workflow while enabling local responses during execution. Precompiled branches reduce the need to synthesize every repair after a violation, but they remain conditional on the failure modes and constraints they represent; unmodeled situations still require diagnosis or escalation.

A corrective action completes recovery only when its result supports re-entry. The Harness reassesses the available evidence and obtains an updated observation or a valid state estimate where needed. This reassessment determines which prior outcomes remain valid. Before resuming, the runtime reconciles assignments and resource commitments and checks the entry conditions of the chosen continuation. Execution may resume within the same subtask, from another valid stage, or along a revised branch. PhyAgentOS records a replan as a child session while preserving the original attempt and its evidence [56]. Both outcomes remain available for review, so later success does not erase the earlier failure.

Recovery requires a feasible continuation with satisfied prerequisites. Where none can be established, the runtime must suspend execution or request human assistance.

Recovery demands also differ from nominal task execution. LIBERO-RECOVER compares responses to resulting failure states in simulation, distinguishing retry and adaptation from object-state and environmental recovery [85]. Section 9.2 discusses what this protocol reveals and the limits of its evidence.

The reviewed mechanisms differ in the task state they retain and the scope they can revise. Assignment methods change responsibility, handoff methods assess successor readiness, and workflow repair preserves or replaces task structure. These differences explain why success on a single capability call is insufficient to assess orchestration. The next section examines the information and computation that support these decisions.

4 Digital Resources in a Physical Agentic Harness

Digital Resources provide information and computation for decisions made during a physical task. We use *Digital Resource* for maintained information, a predictive model or simulator, an executable digital function, or computational capacity that the Harness can access and use. Robot knowledge infrastructures and work on externalized agent resources illustrate support beyond a model's current input [86, 45]. These roles can overlap: a map can answer a query, initialize a simulator, or supply a planner.

We compare resources through the decisions their interfaces support. Their internal algorithms matter when they change what the runtime can query or what evidence a result supplies. Resource selection and use define the Harness role in this comparison. Sections 4.1–4.3 cover knowing the situation, predicting outcomes, and computing solutions. Compute infrastructure supports all three and is discussed with resource selection and execution in Section 4.4. Figure 4 summarizes resource types and their lifecycle. Task Orchestration uses their results to manage the workflow; authorization of operations with physical consequences remains an Execution Governance responsibility.

4.1 Digital Resources for Knowing the Situation

State Representation and Maintenance Maintained state lets later decisions query information beyond the current model input. Its organization determines which questions can be answered efficiently. KnowRob combines semantic knowledge and inference with access to robot and external data, allowing task-specific information to be derived on demand [86]. Three-dimensional dynamic scene graphs instead organize spatial relationships among objects, agents, places, and rooms [87]. Hydra incrementally constructs such a hierarchy, separating rapid local updates from room inference and corrections after revisiting a place [88].

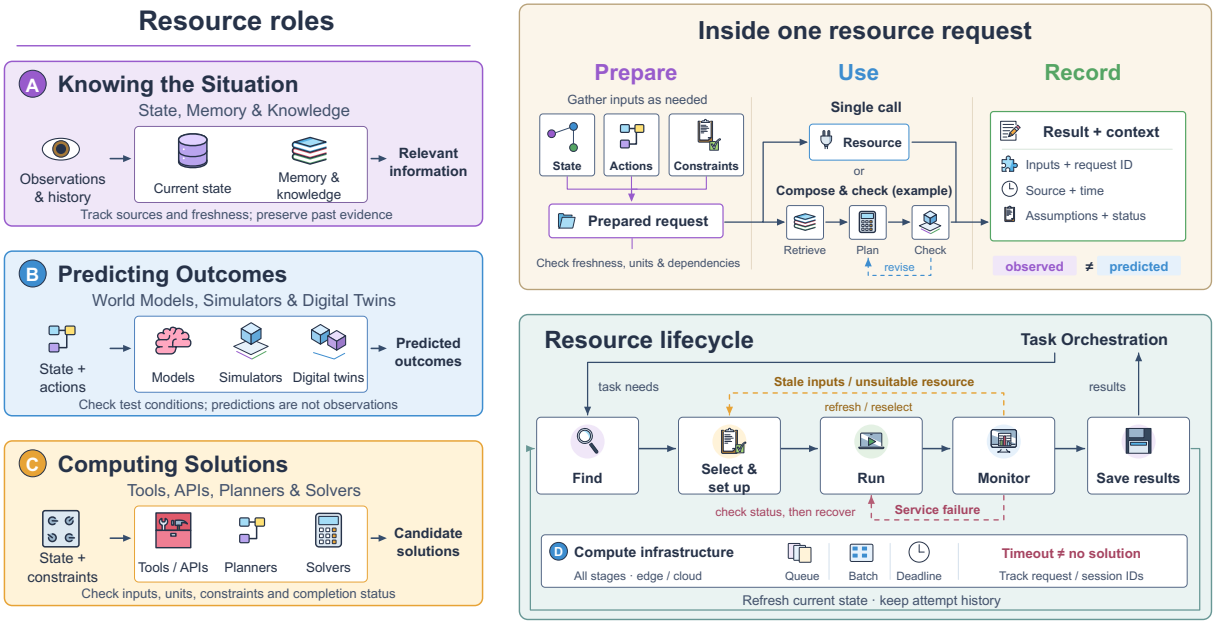


Figure 4 Digital resources in a Physical Agentic Harness. Maintained information (A), predictive models and simulators (B), and computational tools (C) support task decisions. The shared lifecycle coordinates resource use and evidence retention, with feedback guiding resource reselection and service recovery.

The runtime consequence is a difference in how a query is grounded and refreshed. A spatial hierarchy lets a request address an object or room without processing every past sensor measurement. KnowRob-style inference can derive an answer from several records and their relations. That answer depends on its sources even if it is not stored as a separate fact. Hydra's incremental updates instead expose how a maintained spatial representation changes as observations arrive. In either case, a later query may need new evidence, but the affected part of the resource is different.

Current estimates and historical records require different update rules. Thea separates resident context, refreshed observations, and accumulated task records; source and freshness information accompany its scene representation [89]. PhyAgentOS preserves attempts in inspectable state and conditions knowledge consolidation on assessed outcomes [56]. The former controls what enters each model invocation, while the latter also separates an attempted operation from knowledge accepted for reuse.

A new observation can replace an old estimate of an object's location without erasing the evidence that supported an earlier action. Conversely, a historical execution record remains useful for diagnosis even when it no longer describes the current scene. ROSBag MCP makes historical robot data accessible through tools for querying and analyzing recorded ROS messages [90]. Retrieval must preserve whether a record describes current conditions or past behavior.

How state is refreshed also affects the scope of replanning. ADM-Planner separates relatively stable workspace knowledge from changing object state and retrieves a bounded context for the planner [91]. An observation can then invalidate the remaining plan without requiring the system to rebuild all workspace knowledge. Its large simulation study uses a policy surrogate; live language-model tests and physical demonstrations provide a narrower check of the complete planning loop. This separates evidence about memory maintenance from evidence about the deployed model's decisions.

Memory, Provenance, and Evidence Write-back Memory methods trade representational detail against the cost of retrieving useful context. RoboMemory separates spatial, temporal, episodic, and semantic information so that each store can be updated and queried for its own role [92]. HELM takes a

narrower approach around a frozen VLA: it retrieves visually relevant keyframes from the ongoing episode to guide long-horizon control [93]. Its experiments show diminishing returns from adding more retrieved frames. Hierarchical scene representations reduce a different cost by retaining objects and relationships after detailed metric data are discarded [87].

These choices also retain different kinds of evidence. A retrieved keyframe preserves visual context, while an object-level representation makes relations easier to query but may discard details needed for a later visual judgment. Selecting relevant evidence remains necessary as the store grows; HELM's results concern that selection within an episode.

Execution feedback changes which records remain useful. HELM associates episodic memory with verification and recovery, whereas PhyAgentOS gates consolidation using outcomes [93, 56]. A keyframe may help resume the current task without changing support for a later task. Retained lessons or revised retrieval rules raise the separate adoption and improvement questions examined in Section 8.

Memory comparisons also need to control what can be observed after retrieval. MeMoREval supplies a common prior experience, then evaluates memory-conditioned answers and action plans without new observations during the request [94]. This isolates memory modality more directly than a full feedback loop, although its spatial records use simulator-derived information and its action plans cannot repair themselves during execution. SERA instead retains revised planning guidance around frozen planning and action models, with the selected harness evaluated on new instances of the same tasks [39]. Its runtime history summary is updated within an episode and reset between episodes. The two settings test different properties: recalling earlier evidence and using retained guidance during renewed interaction.

4.2 Digital Resources for Predicting Outcomes

Predictive resources examine candidate operations before physical execution. Learned world models estimate future states, simulators provide repeatable interaction environments, and digital twins relate a model to a particular apparatus. The distinction concerns how a test is constructed and what its result represents.

Learned World Models Learned world models support action comparison, but they expose different decisions to the surrounding runtime. DINO-WM predicts visual features under candidate actions and optimizes action sequences without reconstructing complete future images [95]. WorMI retrieves and aligns replaceable world models to the dynamics encountered at test time [96]. These studies expose two runtime choices: searching actions within a predictive model and selecting the model for that search.

Their usefulness depends on which physical effects the prediction represents. A feature prediction may be sufficient to compare candidate motions without reconstructing an image. It may be insufficient for a question that depends on a contact or force effect absent from that representation. Similarly, selecting a model for familiar dynamics does not establish that its prediction remains applicable after the scene changes. The Harness supplies the initial state and candidate actions and can revise its model selection as new observations arrive; the selected model computes the prediction.

Simulators and Digital Twins Explicit simulators make experiments repeatable through controlled initialization and reset. Habitat combines scenes, sensors, and task interfaces for embodied experiments [97]. MagicSim builds executable worlds from declarative specifications and supports batched interaction, realizing high-level commands through controllers, skills, and planners [98]. Their test interfaces and execution semantics make these environments reusable across experiments.

Controlled initialization permits candidates to be compared from a common state, while reset makes repeated trials possible. Batched execution changes the cost of exploring alternatives. MagicSim's use of controllers and skills also matters to the interpretation of a test: issuing a high-level command is different from directly editing the simulated state. A resource used for planning or regression testing therefore needs an execution interface whose effects can be related to the operation under study.

Generative simulation expands the range of test conditions through learned visual dynamics. UniSim learns action-conditioned video generation from heterogeneous data, while Veo-based evaluation generates multi-view video under varied scene conditions [99, 100]. These methods reduce reliance on manually constructing

every environment. They provide a different test substrate from an explicit simulator, where state transitions and object properties are directly represented.

Digital twins and procedural models offer more specific control over the represented apparatus. Alita uses a semantic-enhanced, multi-layer digital twin with feedback interfaces for manipulation replanning and human control [101]. Procedura generates editable assemblies with named parts and machine-checkable relations, then checks connectivity and articulation before producing simulation assets [102]. Alita links a maintained model to operation; Procedura makes the construction and validation of an asset explicit. Both add structure needed to reason about movable parts and their constraints.

Maintained twins can also correct their predicted state using observations. GaussTwin combines physics-based prediction of rigid and deformable object states with visual correction against camera images [103]. The correction uses calibrated camera configurations to update the simulated objects. This updates the represented scene, while calibration supplies the reference relationships needed to interpret the images.

Resource selection depends on the test being constructed: controlled rollouts require an interaction interface, visual predictions require checks on their physical consistency, and apparatus-specific tests require an appropriate model and assets. Procedura contributes to asset preparation; governing execution requires additional runtime mechanisms.

SpatialHarness maintains a synchronized spatial scene and supplies selected virtual views and object-pose information for fine manipulation [104]. Its four-task Franka comparison improves completion from 11/60 to 50/60 trials under the augmented interface. This supports the tested spatial-support package, which includes scene construction and additional information; policy-query counts alone do not measure its total cost. Auriga uses candidate-motion previews and execution receipts to connect proposed manipulation programs to geometric and observed evidence [105]. Its component ablations support particular checks on selected simulated cases. The RoboCasa evaluation supplies rendered-depth maps and task-success predicates, while its RoboDojo setting withholds simulator depth. These information conditions and paused simulation limit conclusions about real-time hardware operation.

Interpreting Predictive Evidence A prediction supports a decision only within the conditions represented by its model. Initial state, dynamics, and test duration determine the scope of a simulated result. For generated video, visual plausibility can coexist with inconsistent contact, views, or long-term state [99, 100]. Assessing these resources requires identifying the modeled effects and checking their assumptions against current observations.

4.3 Digital Resources for Computing Solutions

Computational resources turn task information into candidate solutions. Direct calls, generated programs, and constraint-based search expose different intermediate results and failure signals.

Tool Invocation and External APIs A direct tool call delegates a bounded computation to an external implementation. Toolformer studies learned decisions about when and how to call tools, while ReAct uses returned observations to revise subsequent reasoning and actions [106, 107]. These digital-agent studies provide methodological precedents for resource access. In a physical task, either pattern can retrieve a record or request a route, but the caller still needs to interpret the result before acting on it.

The tool's completion status determines what that result establishes. A planner that establishes infeasibility under the supplied constraints gives a reason to change the candidate or its assumptions. A timeout leaves the planning question unresolved; it can motivate another computation without showing that the physical operation is infeasible. Retaining both the returned value and completion status lets the caller distinguish an infeasible candidate from an unresolved computation.

Programmatic Composition Generated programs make dependencies among calls explicit. Code as Policies combines perception outputs, geometric computations, and exposed control functions through code with branches, loops, and generated helpers [32]. The program exposes how an intermediate value affects later calls, whereas a sequence of independent invocations leaves that relation to the caller. Programs can

mix digital computations with physical skills; the function invoked determines which execution conditions apply.

The available primitives and feedback determine how a coding agent can revise that program. CaP-X compares action abstractions and feedback configurations, showing that the code interface itself affects manipulation performance [108]. Agent as Policy uses live programs and physical feedback to revise behavior during real-robot manipulation [109]. Its trials permit recovery within fixed time and interaction budgets, while human intervention ends an autonomous trial.

Constraint-based composition adds an explicit search criterion. PRoC3S searches continuous program parameters and uses constraint violations to guide revision [110]. Meta-Optimization and Program Search combines language-guided constraint selection with parameter and trajectory optimization [111]. SayPlan instead connects hierarchical scene-graph search to path planning and simulation checks [112]. Their interfaces determine which part of a candidate can be revised after a failed check.

For the surrounding runtime, these representations help determine the scope of a correction. In parameterized programs, a failed constraint can motivate a new continuous parameter while leaving the program structure intact. A missing operation or unsuitable dependency instead requires a structural change. The solver performs the search within its encoded problem. The Harness can retain its result, supply revised inputs, or request a different computation when the problem assumptions no longer apply.

Solver Verification and Reusable Results Solver verification is bounded by the encoded problem and its assumptions. PRoC3S makes constraint satisfaction central to candidate search, while SayPlan checks a plan against its scene representation [110, 112]. If that representation is outdated, successful computation can still yield an unsuitable physical proposal. For comparison across methods, the relevant questions are which assumptions are explicit and which intermediate failure can trigger a local revision.

Reusing such a result also requires knowing what has changed since it was computed. A route can cease to apply after an obstacle moves even though the planner returned normally. An intermediate geometric calculation may remain usable if only a later parameter search failed. As a synthesis of the methods above, preserving the relation between inputs, checks, and returned status supports selective recomputation. This allows the runtime to identify which computation needs to be repeated, with record formats adapted to each tool.

4.4 Using Digital Resources: Selection, Execution, and Feedback

Selection and Preparation Resource retrieval narrows the support exposed to a model before requests are executed. ToolLLM retrieves candidate tools from API documentation, while scalable cloud tool access uses a semantic–lexical gateway to select a bounded set and preserve session routing [113, 114]. Task-aware provisioning additionally expands a task-specific configuration after a failed self-check [115]. These digital-service studies differ in whether selection is a one-time retrieval step or a decision revisited during execution.

Descriptions can also capture applicability beyond an API signature. RoboHarness uses policy cards and policy-specific memory to describe robot policies and their outcomes [116]. The policies are embodied capabilities, but their searchable descriptions illustrate the resource needed to compare candidates. Selection can be revisited when the current decision requires inputs or evidence that the chosen resource cannot supply.

Selection is followed by preparation of a particular request. For retrieval, this can mean choosing a compact state view; for simulation, it means supplying an initial state and candidate actions. A generated program additionally depends on the tools and libraries available to its calls. Missing inputs require further retrieval, and an unavailable implementation may require another resource. These preparation failures leave the original computation unresolved.

Compute Allocation and Serving Compute placement determines when a selected resource becomes usable. RILaaS exposes authenticated model services and distributes requests across edge and cloud servers under changing load [117]. FogROS2 moves selected ROS 2 computations to cloud resources through deployment and networking support [118]. RoboECC instead partitions a model between edge and cloud and adapts

the split to bandwidth [119]. The first two methods place whole requests or services; RoboECC changes the division of computation within one model.

A remote processor can reduce inference time while increasing the time needed to obtain a usable answer. Input transfer, queuing, and result delivery remain on the request path. RoboECC's bandwidth-dependent partitioning responds to this coupling within a model, whereas service placement changes where a larger computation runs. The comparison is therefore sensitive to workload and communication conditions. Placing a slow planning request remotely also does not imply that a time-critical controller can use the same execution arrangement.

Scheduling within a processor addresses a different bottleneck from service placement. KerColle overlaps GPU work from concurrent VLA requests at kernel level [120]. Its GPU-architecture simulations measure compute throughput; assessing closed-loop robot response would also require checking whether returned actions arrive before their supporting observations become stale.

Robot-aware scheduling adds the timing of physical execution to this compute problem. Kairos adjusts executed action horizons using inference confidence and schedules robots with attention to computation-induced waiting [121]. The scheduling objective accounts for when the next result remains useful to the robot. Batching and action-horizon selection can consequently affect each other even when the model and hardware are unchanged.

Shared serving systems separate model-specific behavior from reusable execution support. PhyAI retains model-specific input, solver, cache, and output logic above shared kernels and memory-management services [122]. Z-Infra separates environment and model workers, batches compatible requests, bounds queued work, and returns asynchronous results through session identifiers [37]. The former emphasizes support for heterogeneous model implementations; the latter makes concurrency and session lifecycle explicit.

Service Monitoring and Feedback Failure handling depends on what the serving system can observe. RILaaS uses monitoring to adapt request distribution, whereas Z-Infra uses leases, heartbeats, and lost-session notifications to expose worker and session failures [117, 37]. Load imbalance can justify redistribution, but a lost session first leaves the status of its work unresolved.

The response also depends on whether the original work can still be identified. A worker failure may leave an outstanding session whose completion is unknown, while overload can leave a valid request waiting in a queue. Rerouting must preserve the association between the returned result, the request, and its inputs. Otherwise a late result from the original worker can be mistaken for evidence about a replacement request.

Across these methods, resource quality and service availability are separate sources of failure. Another server may resolve an unavailable computation without correcting stale inputs or an unsuitable predictive model. A mismatch in inputs or model applicability calls for resource reselection; a service fault calls for execution-status handling.

5 Embodied Capabilities in a Physical Agentic Harness

Embodied Capabilities expose callable perception and action implementations for physical tasks. Their operating conditions depend on the robot body, sensors, control interface, and workspace [123, 24]. A runtime description connects these conditions to the task requests an implementation can support [31, 124, 32]. The Harness uses that description to select and prepare a capability.

Figure 5 relates capability implementations to their runtime management. We first distinguish perception and action interfaces and compare representative action implementations, then examine capability management, composition, and adaptation.

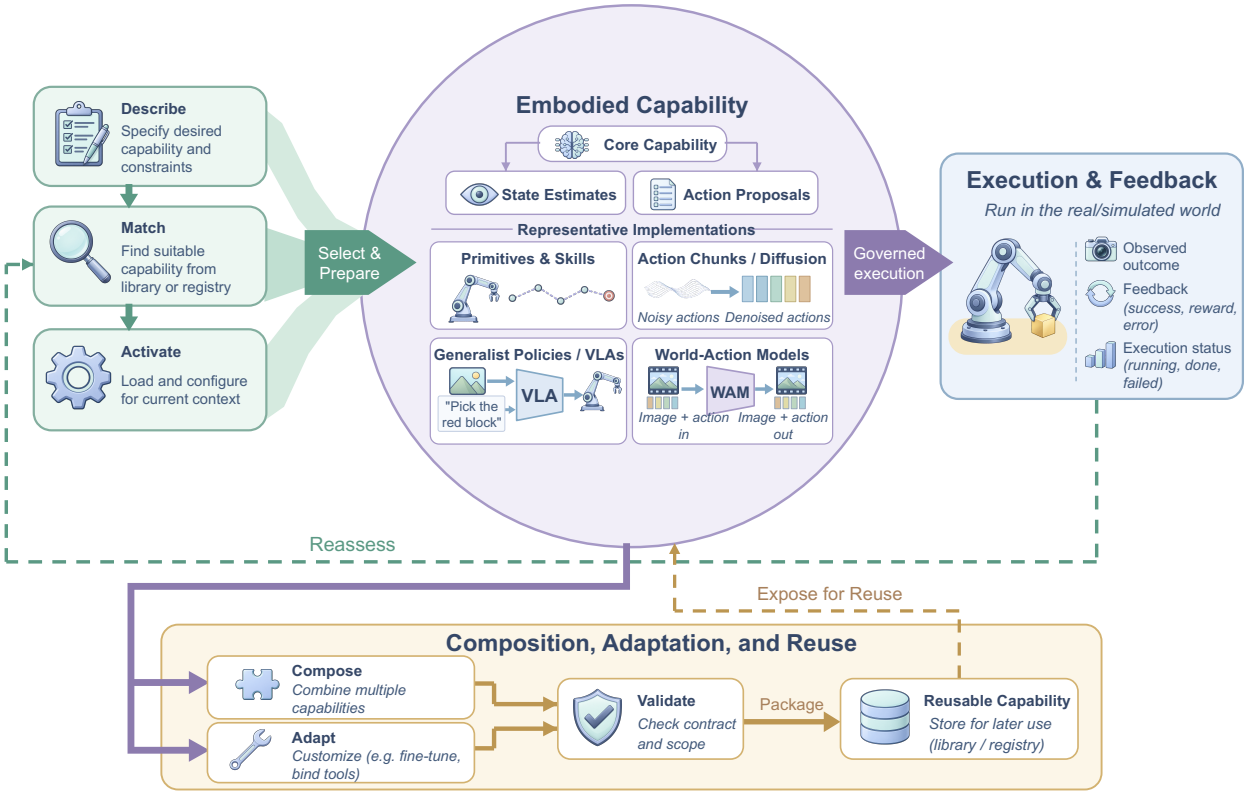


Figure 5 Overview of embodied capabilities in a Physical Agentic Harness. Callable perception implementations produce task-relevant estimates and affordances; action implementations produce candidate behavior. Representative implementations include analytic primitives, learned policies, generalist VLAs, and world-conditioned action models. These forms can overlap within a system. Runtime management describes, matches, and activates implementations. Composition, adaptation, and boundary conditions determine how capabilities remain usable and reusable.

5.1 Perception and Action Interfaces

An *Embodied Capability* is a callable perception or action function associated with a robot embodiment. Its interface exposes the inputs, outputs, and operating conditions needed to use it. A perceptual capability returns a task-relevant estimate or affordance. An action capability produces behavior for execution, such as a navigation command or manipulation sequence. The distinction follows the exposed function: a grasp detector can return candidate poses or be incorporated into a capability that executes a grasp [125]. A VLA can internally perform perception and grounding while presenting an action interface to the Harness [1, 2].

Perception connects task language to measurable conditions and possible actions. ConceptFusion combines open-set multimodal features with 3D maps, while VLMaps grounds language in a spatial representation [126, 127]. Their outputs let a task refer to an object or location through a representation that other components can query. Affordance methods further relate an observation to an available behavior. AnyGrasp estimates grasp poses from observed geometry [125]. These functions differ from storing the resulting estimates, which is a Digital Resources role.

Using a language-grounded target for grasping requires selecting a grasp candidate appropriate for both that target and the robot. OK-Robot makes this integration concrete by joining open-vocabulary perception, navigation, and grasping in household tasks [128]. Its composition illustrates why the interface between estimates and actions matters alongside the quality of each model.

Robo-Harness K1 makes task-relevant perception directly queryable by a robot-use agent [129]. Its tools expose calibrated depth, persistent visual anchors, spatial measurements, and grasp hypotheses to an existing

VLM. The agent can request geometric evidence when an image alone does not resolve the next action. The principal change is the geometric evidence available to the existing model through its interface. Its simulation comparisons evaluate the resulting robot-use system; they do not isolate every perceptual tool from the accompanying memory and execution feedback.

The usefulness of a perceptual output depends on its context. An object location needs a reference frame, and its acquisition time affects whether it can still support an action. Affordances also depend on the current body and environment rather than being permanent object properties. VoxPoser and SayPlan illustrate how spatial representations and feedback ground task decisions [130, 112]. Uncertainty and validity information help the Harness decide whether an estimate is sufficient or another observation is needed.

An action interface can delegate low-level tracking to a robot-specific controller. ViNT predicts relative navigation waypoints, which the controller tracks on the selected robot [131]. Unlike a perceptual estimate, this output specifies a desired motion.

The separation between waypoint prediction and tracking also limits what can be inferred from a shared model interface. The same type of waypoint output can be used on different robots while each controller realizes it under its own motion constraints. The model output is therefore only one part of the deployed capability. Its operating conditions include the checkpoint, controller, and interpretation of the output.

An interface can also separate preparing a motion from executing it. VIA lets an agent inspect and edit a gripper target in a visual 3D interface before an explicit waypoint-execution call [132]. World Action Agent similarly exposes contact-oriented views and editable action proposals, with planning feedback before execution and visual correction afterward [133]. VIA emphasizes visual target specification, whereas World Action Agent adds rehearsal within a longer manipulation process. Both evaluate these interfaces in simulation, where inspecting a proposal remains distinct from testing its eventual physical effect.

Action interfaces also introduce a temporal commitment. A controller call, an action chunk, and a long-horizon skill expose different opportunities for reassessment. Completion of inference or command delivery is distinct from completion of the intended physical effect. The runtime therefore needs to know what the implementation reports, when it can be interrupted, and which observations support its completion criterion. These questions apply across the implementation families below.

5.2 Action Capability Implementations

Action implementations expose different opportunities for runtime selection and supervision. The comparison below asks what behavior an invocation commits to and when feedback can change that behavior. It also asks which operating conditions are exposed to the caller and which remain implicit in the implementation. We use analytic skills, learned action sequences, generalist policies, and world-conditioned generation to examine these differences. These groups overlap within systems: a generalist policy may produce action chunks and use predicted future states. The action algorithms supply the capabilities; their invocation, execution records, and external checks supply the Harness support discussed here.

Analytic Primitives and Reusable Skills Analytic implementations expose operations such as inverse kinematics, trajectory tracking, or grasp execution. Skill frameworks combine such operations with preconditions, resources, and completion conditions. SkiROS2 and behavior-tree methods provide representations through which a task executive can inspect and compose these behaviors [29, 134]. This structure makes some assumptions explicit before invocation. It does not guarantee success when geometry or contact conditions change, and the interface must distinguish internally checked conditions from those monitored elsewhere.

Preconditions, hold conditions, and postconditions let the executive reason about invocation, continued execution, and outcome at skill level. Skill frameworks and behavior trees make these relationships inspectable without requiring the executive to reproduce every controller update. The controller still generates motion; the surrounding support uses the conditions to select, monitor, or replace the invocation. A failed condition can localize the operation requiring attention and leave unaffected task stages intact.

Learned Action Chunks and Diffusion Policies Learned policies can predict a sequence of future actions while committing only part of it to execution. ACT uses temporal ensembling to combine overlapping action-chunk predictions, while Diffusion Policy generates actions through conditional diffusion and replans after executing a prefix [135, 136]. Combining predictions and choosing the executed interval are separate design choices that can coexist. Neither choice alone specifies how an external runtime can interrupt the controller.

The invocation interface determines which part of this sequence the Harness can still revise. A prediction horizon longer than the executed interval leaves actions that may be replaced after new observations, if the controller exposes that operation. The runtime therefore needs to relate each sequence to its supporting observation and distinguish pending actions from committed execution. Changing the latter may require a controller-level intervention. This distinction also separates failure to produce the next sequence in time from failure of the physical action already underway. The end of a generated sequence still requires a separate assessment of the intended task effect.

Generalist Policies and Vision-Language-Action Models Generalist policies broaden the instructions and tasks supported by one implementation. RT-1 learns multi-task robot control, RT-2 incorporates vision–language pretraining into action generation, and Octo and OpenVLA provide models that can be adapted to particular robot settings [137, 1, 138, 2]. The π_0 family uses flow-based action generation and extends task coverage through heterogeneous training data [3, 139]. For runtime use, the relevant comparison is which observation and action conventions each deployment supports. A policy that accepts many language instructions does not automatically support a new camera configuration, controller, or robot body.

Task selection and deployment selection therefore remain different runtime decisions. An instruction selects behavior within a supported policy interface. Deployment selection requires checking whether a checkpoint and adapter support the current robot. This assessment must account for the available observations and the controller that will interpret the outputs.

An implementation profile supplies the evidence for that deployment choice. It relates input conventions and action interpretation to the tasks and embodiments on which the implementation was evaluated. As a runtime abstraction, it can distinguish several deployments of one checkpoint without assuming that their reliability is identical. The profile supplies runtime information about supported use; the policy implements the learned mapping. If the action-generating foundation model is itself the Base Model, the Harness still provides its surrounding invocation and execution support. A separate task-reasoning model is not required.

Some action models are explicitly designed to receive structured guidance from an agent. 2AM converts agent-side memory into subtask language and spatial hints for an otherwise episode-stateless action model [140]. AR-WAM instead conditions execution on a target box and operation token, leaving longer-horizon memory and recovery to the agent [141]. Both rely on compatible action-model training; their interfaces are not interchangeable additions to any frozen policy.

World-Conditioned Action Generation Some implementations use predicted state evolution directly in generating an action. Predictive approaches such as UniPi connect anticipated visual outcomes to planning, while unified world-model approaches study prediction together with decision making [142, 143]. In this setting, the predicted future is an intermediate representation used to construct behavior. The runtime question is how that representation becomes an executable action for the available body.

Cascaded and joint approaches place the prediction at different points in action generation. A cascade can predict a visual future before decoding actions; a joint model couples future observations and actions within its generative process. GR-2, Unified Video Action Model, and DreamZero provide examples of connecting future-state modeling to action generation [144, 145, 10]. This internal organization does not alone determine what a caller can inspect. An interface that returns only actions gives the Harness different evidence from one that also exposes the predicted trajectory. The exposed outputs determine whether the runtime can use the prediction as evidence.

The functional distinction is how the prediction is used at invocation. A model queried for a counterfactual outcome acts as a Digital Resource. An implementation that turns predicted evolution into robot behavior provides an action capability; one system can expose both roles [146].

When a predicted trajectory is exposed, it can support a comparison with later observations. Its interpretation depends on the conditioning observation, prediction horizon, and mapping to the robot's actions. The surrounding runtime can use a discrepancy to request a new prediction or reassess a pending operation. If the prediction remains internal, that particular check is unavailable through the interface. In either case, execution evidence is still needed to determine what actually occurred.

Across these implementations, capability granularity determines what the Harness can select and supervise. Narrow skills can expose geometric tolerances or controller states; broader learned policies often rely more on empirical applicability conditions. Hierarchical systems combine several granularities rather than choosing one for every task [13]. Capability descriptions can preserve these differences while supporting substitution at a common task interface [147, 148].

The management problem that follows is to preserve these interface differences during reuse. A model can be exposed through several task-specific capabilities with different goals and termination tests. Conversely, a longer capability can combine learned behavior with explicit control primitives. The same model family can therefore support different execution units. Selecting a unit also selects its feedback and intervention conditions. Capability management records those conditions for each concrete request.

Recent whole-body and corrective interfaces make the role of training explicit. HITCH connects language-model proposals to whole-body planning and a learned tracking controller; its real G1 tests use a trained vision-language model and tracker [149]. A2-MEAS combines an action model with a learned correction interface and bounded diagnostic retries [150]. Their results depend on the trained interfaces used in each evaluation. TinyAstral similarly distinguishes improvements from a revised harness around fixed models from the performance of a student subsequently trained on harness-mediated trajectories [151]. These designs expose capability contracts worth comparing while requiring separate accounting for interface engineering and model training.

5.3 Capability Management

Capability management connects an available implementation to a concrete invocation. Description records supported conditions, matching assesses the current request, and activation prepares an instance. Embodied-Skills links executable skills to prerequisite checks, outcome verification, and recorded execution events [65]. Such mechanisms make a capability inspectable across use, although individual systems may combine the management stages.

Describing Capabilities as Runtime Contracts A capability description relates an API to its physical operating assumptions. Industrial Robot Skills and SkiROS2 describe typed parameters, resources, and procedural conditions. AEROS and ECM Contracts extend runtime descriptions to compatibility, recovery, and governance concerns [50, 29, 152, 148]. Their schemas cover different subsets of these conditions.

Stable implementation metadata also differs from invocation state. Supported sensors, action parameterization, and software dependencies describe an implementation. Object bindings, current observations, and available resources describe a particular use. Keeping that distinction permits an observation update without redefining the capability. It also helps compare implementations that satisfy the same task request through different adapters or control assumptions.

Observable status gives this description meaning during execution. A timeout leaves the intended effect unresolved, whereas a verified failure supplies evidence that the effect was not achieved under the tested condition. An interruption can leave a partially completed operation whose outputs are still relevant. These distinctions connect the contract to the physical state left by a call. They are especially important when substituting implementations: identical function names do not ensure identical stopping behavior or completion evidence.

Matching Capabilities under Embodiment Constraints Matching relates task relevance to present feasibility. ProgPrompt conditions planning on available actions and objects, while SayCan combines language relevance with estimates of skill affordance [124, 31]. Feedback and scene representations provide further evidence about whether a candidate applies now [14, 130, 112]. A missing required sensor or incompatible action format can exclude an implementation. Physical prerequisites still need to be satisfied even when a

capability is highly relevant to the instruction. Expected latency or past performance can rank candidates that remain feasible.

Incomplete evidence creates a different result from established incompatibility. An occluded target may require additional perception rather than immediate rejection of the capability. Conditions can also change between matching and invocation. Matching can therefore be revisited when an observation or resource assumption changes.

Activating and Instantiating Capabilities Activation prepares the selected implementation for a concrete request. It grounds arguments and configures the required resources and feedback channels. Generated programs in Code as Policies and feedback-based execution in Inner Monologue illustrate ways to connect selected functions to task-specific inputs [32, 14]. Device discovery and binding can occur during this preparation. We call the information describing the prepared operation a *capability proposal*; its representation can vary across implementations. Execution Governance assesses physical dispatch using that information and the applicable constraints.

Grounding connects this preparation to a specific execution. An abstract object reference must resolve to the target used by the selected implementation, and a pose must be interpreted through the active observation and action adapters. Code as Policies exposes some of these dependencies in generated calls, while Inner Monologue uses feedback to revise the context in which actions are selected. Retaining the implementation and grounding context lets later status be attributed to that instance. Without this association, a correct result from an earlier invocation can be confused with completion of the current request.

Preparation can fail without any physical action occurring. An unavailable model service or incompatible adapter is therefore different from a failed grasp. After dispatch, feedback must refer to the configuration that produced the operation. If a change of implementation, target, or adapter invalidates admission assumptions, the affected proposal requires reassessment.

5.4 Capability Composition and Adaptation

Composition connects functions through both their outputs and their physical operating conditions. OK-Robot integrates recognition, navigation, and grasping, while Thea organizes callable tools around physical feedback [128, 89]. Harness VLA combines a frozen action policy with external support for contact-rich execution, and RoboBRIDGE adds adapters, monitoring, and replanning around pretrained policies [26, 82]. These systems illustrate different levels at which a capability can be composed with other mechanisms.

Composition also depends on the successor conditions discussed in Section 3.2. A reusable capability interface exposes those conditions so that orchestration can assess them against the state produced by the previous call.

Intermediate outputs determine how much of a composition can be reused after failure. A grasp failure does not necessarily invalidate the map used to reach the object. A changed object pose can, however, invalidate several downstream motions grounded in the previous estimate. This is the capability-side reason to preserve dependencies between outputs and their consumers. Task-level recovery can use these dependencies to retain valid results and repeat only the affected invocations.

Adaptation changes an implementation or its interface for a deployment. Adapters translate conventions across platforms; separately managed policies can be specialized to new observations or actions, as illustrated by Octo and OpenVLA [138, 2]. Calibration changes instead affect the references maintained through Digital–Physical Interaction. These adjustments can support reuse, but their validity depends on the tested body and conditions. Preparing an existing implementation for the current task differs from retaining a revised component for later episodes.

Interface adaptation and policy specialization also change different assumptions. An adapter can translate a supported observation or action convention without altering the policy weights. Fine-tuning changes the learned behavior and therefore requires evidence for the revised deployment. Policy reuse therefore requires empirical validation of the adapted deployment, as illustrated by Octo and OpenVLA. If an adaptation

changes the conditions exposed by the capability, its callers may need to be reassessed even when the task-level name remains unchanged.

Reusable compositions can themselves enter a capability library. ASPIRE repairs and validates code-as-policy skills, while EmbodiSkill distinguishes a defect in stored skill guidance from a lapse during execution before applying revisions [153, 154]. These mechanisms connect composition to Persistent Evolution. Their reuse still depends on preserving the constituent assumptions and the scope of the validation. The next section examines how proposed behavior is admitted and supervised once such capabilities are invoked.

6 Execution Governance in a Physical Agentic Harness

Execution Governance decides whether and under what constraints a physical action may proceed. It also determines whether the observed outcome meets the criteria for acceptance. Task and capability context, applicable constraints, and execution evidence support both decisions. Task Orchestration uses the resulting status to manage the workflow, while Digital–Physical Interaction supplies device-grounded observations and carries admitted commands.

The candidate may be a tool call, an action chunk, a policy command, or an executable program. Governance applies at the granularity at which the runtime can inspect a candidate and enforce the resulting decision on its physical execution, so admitting a program does not imply that every internal step has the same gate. The relevant context may include a capability contract, target and parameters, preconditions, current body or environment state, live telemetry, authority state, or an outcome record. A capability catalog, fixed serializer, or transport bridge may support this path. The governance function is the decision that constrains execution or assesses an outcome for acceptance. Before or during execution, such decisions can allow, modify, or block an action [155, 156, 157]. Some runtimes also defer or stop an action when conditions remain unresolved [155, 158].

Governance can be analyzed around three decision points in the action lifecycle. Before execution, it checks whether a proposed action may start. During execution, it monitors the action and can intervene or transfer control. At closure, it determines whether the observed outcome satisfies the acceptance criteria. Audit fields may be written throughout the lifecycle and used to support closure and subsequent decisions. These decision points can be combined within a runtime and revisited as execution changes [30, 15, 158]. A system may connect them through a shared execution record.

6.1 Governance State and Executable Constraints

Governance state is the information that a runtime can inspect when deciding about a physical action. It may include the requested capability and arguments, target and intended effect, current body and environment state, preconditions, physical limits, permissions, freshness, uncertainty, authority, and available recovery options. A check becomes operational when its result constrains execution or determines whether an observed outcome is accepted. Before or during execution, it can permit, modify, defer, or block an operation; at closure, it can leave an outcome unresolved until the required evidence is available. A diagnostic contributes to governance when a runtime decision consumes it.

Some systems expose governance state through the tool interface. ROSClaw gives its validator a typed manifest, allowlist, velocity bounds, and normalized observations before a ROS call is published [156]. LabGuard instead compiles laboratory rules into an executable representation and checks live chemical and kinematic state [157].

Other representations make authority and evidence obligations explicit. ETA keeps tool identity, side effects, fresh-observation obligations, and trusted receipts under host control [155]. The runtime-governance framework also represents telemetry, intervention state, and recovery conditions [158]. These fields connect an individual check to the conditions under which execution may continue.

These representations differ in how much they depend on current evidence. Static type and permission rules can govern if an online decision consumes their result. State-dependent checks may additionally require a current observation, a valid endpoint, a non-stale execution context, or a feasible recovery condition. Keeping

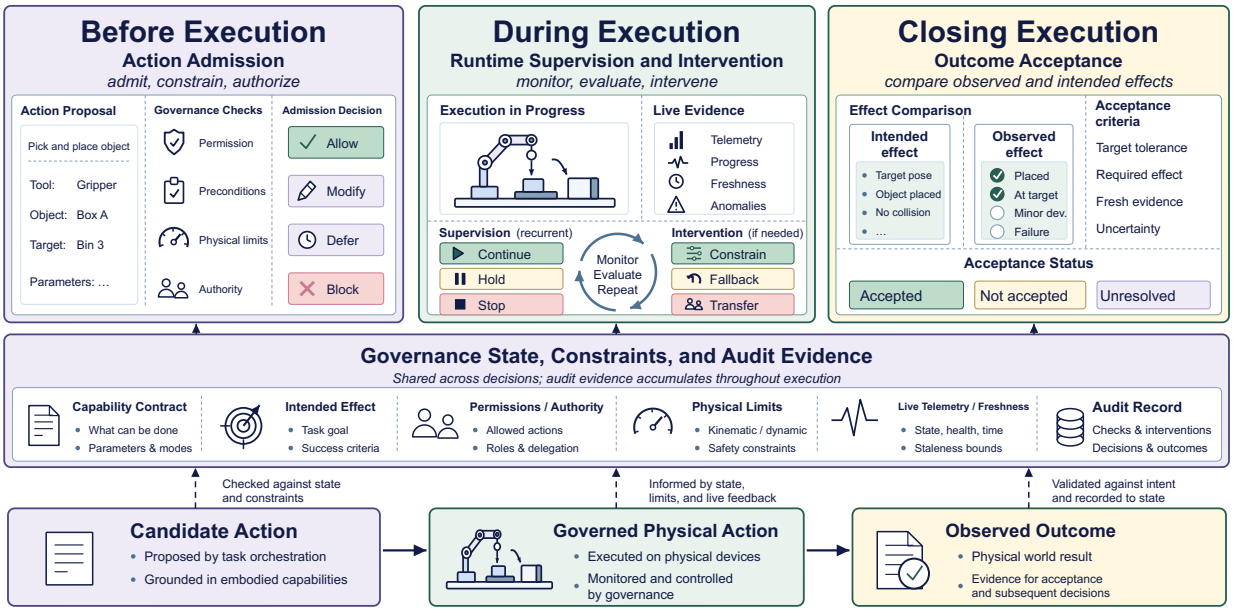


Figure 6 Execution Governance organizes a physical action around three decision points. Before execution, action admission determines whether and how a candidate starts. During execution, supervision reads live evidence and intervention changes the running action or transfers control. After execution, outcome acceptance compares intended and observed effects and preserves audit evidence. A shared governance state supplies the capability contract, intent, authority, physical limits, live state, and recovery options used across these decisions. The arrows indicate analytical relations rather than a mandatory one-way execution sequence.

these cases separate lets the survey distinguish syntactic correctness, physical feasibility, policy compliance, and human authorization. The comparison here concerns the information consumed by a check and its effect on physical execution, where the source reports those details.

6.2 Before Execution: Action Admission

Action admission assesses a candidate before physical dispatch. The checks and the resulting decision are distinct: a candidate may pass some interface checks while still lacking permission or sufficient evidence to start.

Permission and Constraint Checks Admission checks use this state to test whether a particular request can proceed. ROSClaw checks a call against configured command restrictions before publication, while LabGuard evaluates its permissibility under the current laboratory state before dispatch [156, 157]. A request that still conforms to its interface can fail a state-dependent check after the environment changes.

Admission can also depend on whether the evidence required by a check is available. ETA prevents a new world-changing call while fresh-observation obligations remain unresolved [155]. The runtime-governance framework relates capability metadata to policy context and current telemetry [158]. These checks distinguish a request that violates a known constraint from one whose permissibility cannot yet be established. The following decisions determine how each case affects dispatch.

SafeHarness separates route admission from contact execution, checking obstacle geometry and carried-object clearance before motion [41]. Its SafeLIBERO experiments report task success, collision avoidance, and their conjunction separately. In the matched GPT-5.5 comparison, safe success increases from 31.3% with model-and-skills support to 73.1% with the full harness over 320 simulated episodes. The collision proxy is obstacle displacement, so this result does not certify all contact hazards or model errors. CarryOn shows why violation

reduction must be assessed alongside completion: in its LIBERO-Safety comparison using a safety-fine-tuned $\pi_{0.5}$ policy, violations decrease from 52.6% to 6.1%, while violation-free completion changes only from 25.3% to 25.6% [159].

Admission Decisions and Escalation The result of a check determines how the proposed operation changes. ROSClaw allows conforming calls or blocks them with a structured rationale. LabGuard's SafeFilter additionally supports modifying an action before execution [156, 157]. Admission mechanisms therefore vary in whether they reject a request or permit a modified action.

Other mechanisms preserve an unresolved decision. ETA blocks further dispatch until observation obligations are discharged and can force a safe stop [155]. The runtime-governance framework defines accept, reject, defer, and escalate outcomes [158]. Deferral leaves a condition to be resolved; escalation transfers the decision to an additional review path. The existence of that interface should be distinguished from a demonstrated live takeover.

6.3 During Execution: Supervision and Intervention

Runtime Supervision Once an action has started, *runtime supervision* maintains an active execution state, reads telemetry or progress signals, and evaluates whether the current mode remains valid. Missing, stale, or delayed evidence becomes governance-relevant when it triggers a control decision about continuation, pause, stop, or recovery.

Controller-level supervision monitors whether a safe execution mode remains available. SOTER uses safe regions and reachability conditions to decide when an advanced controller should no longer remain active [30]. AESOP addresses the additional delay of language-model reasoning. Its fast anomaly monitor triggers slower intervention selection while a receding-horizon controller preserves feasible recovery options [15]. The formal result depends on the modeled dynamics, recovery-set invariance, and latency bound.

Task-relevant supervision uses a different level of state. RoboBRIDGE tracks the active primitive, perception, and monitor confidence to detect failures that require stopping and diagnosis [82]. PhyAgentOS uses session state, heartbeats, and verification to identify stalled or semantically failed execution [56]. These checks can expose failures that a low-level controller does not represent. Their effectiveness still depends on what the monitor can observe and which physical interventions it can request.

Supervision can also check a physical prerequisite before permitting a dependent action. CommitFlow verifies semantic commitments during VLA execution and applies local correction when a prerequisite is unmet [160]. It rechecks the corrected state and preserves unaffected progress. The base policy remains frozen, but the verifier and correction components are trained. Simulation and hardware trials evaluate whether this check-and-correct loop enables the original policy to continue.

The Physical Agentic Loop illustrates a narrower, physical-task-specific version of the same pattern. A Watchdog maps gripper effort, closure, micro-lift, and timeout signals to structured outcome labels, and a bounded policy uses the label and attempt history to finalize, retry, reselect, ask for clarification, or terminate. Its Stretch experiments provide direct hardware evidence for this execution-state path, with automatic retry exercised for selected failure classes. The evidence concerns monitorable, bounded recovery around one grasp primitive [161].

BT-Agent represents execution through behavior trees whose preconditions and periodic guards can stop action independently of the language-model actor [162]. Its physical Spot trials retain operator approval for motion. The reported median 19 ms intervention interval measures issuance of a zero-velocity command after local acceptance, not the time until the robot physically stops. FastHarness changes review frequency using retained experience, reducing language-model supervision on familiar stages while retaining error-triggered review and audits [163]. Its four-task simulation study supports a review-cost tradeoff, but missed failures and the short evaluation sequences preclude a claim of reliable long-term supervision.

Intervention and Control Transfer Execution-time intervention applies an explicit control effect to the active physical action or to the authority that controls it. It may constrain, pause, hold the robot's position, stop, replace the active command, or transfer the active control path. Updating workflow progress alone is not an execution-time intervention. Outcome acceptance is a separate governance decision, discussed in Section 6.4. The scope of intervention authority must be stated explicitly. A Harness may veto or redirect a policy action while a downstream controller, hardware interlock, or independent emergency stop retains final actuator authority.

Controller switching and task-level recovery affect execution through different mechanisms. SOTER disables the advanced-controller output path and permits its return only after a recovery condition is reached [30]. AESOP selects among bounded recovery options while accounting for the time needed to choose one [15]. Their guarantees depend on the controller and timing assumptions used in the analysis.

RoboBRIDGE and PhyAgentOS connect stopping or cancellation to broader recovery decisions [82, 56]. Stopping or cancellation changes the active action or its authority; orchestration then selects the task continuation. A stop request also depends on downstream control semantics; issuing it does not establish an instantaneous physical halt.

MIRA separates rapid interruption from the slower interpretation of a person's new request [164]. Its full-duplex interaction system limits committed behavior to a short prefix and uses a fast speech-triggered interruption path alongside deliberative turn arbitration. This lets a robot respond to an interruption before deciding how the task should continue. The reported interruption latency ends at abort-command dispatch, so it must not be interpreted as the time until physical motion stops.

Transfer to a human or another service changes who may issue or hold execution authority. The runtime-governance framework defines escalation and takeover interfaces. ROSClaw's independent "/estop" path illustrates stop authority, not authority transfer. An implemented transfer interface and an exercised live takeover-and-return trial should therefore be reported separately [158, 156].

6.4 Closing Execution: Outcome Acceptance and Audit

An interface-level success status does not necessarily establish that the intended physical effect occurred. *Outcome acceptance* is the runtime decision that combines the intended effect, current or refreshed observations, device- or environment-native signals, and explicit acceptance criteria. An *audit record* preserves the proposal, checks, interventions, and outcome as evidence used to close the current action and support subsequent decisions. Acceptance of an action outcome informs task progress; Task Orchestration determines whether that outcome satisfies the dependencies of the wider workflow. The audit record contributes to governance when a decision uses it to constrain execution or assess the outcome for acceptance.

Outcome checks differ in the evidence they use and in what follows an unresolved result. ETA combines a host-attested receipt with fresh observation obligations that can block the next world-changing call [155]. The Physical Agentic Loop uses telemetry and attempt history to assess a grasp, while PhyAgentOS evaluates the session objective against refreshed state and action–observation history [161, 56]. A PhyAgentOS replan verdict creates a child session, preserving the original attempt. ROSClaw's append-only records instead support within-trial diagnosis by linking observations, calls, admission decisions, and outcomes [156]. Their evidential scope ranges from physical-outcome assessment to diagnosis of the execution trace.

RegenHarness makes the separation between observation and accepted progress explicit [165]. Its commit gate checks evidence associated with the current task and configuration before advancing accepted state. An unresolved result remains uncommitted and can trigger bounded recovery. The mechanism addresses premature acceptance of progress; it still depends on the underlying observations being sufficient to establish the physical effect.

The evaluator itself also requires validation. RoboValue tests learned execution judgments using instruction changes, forward/reverse trajectory replay, recurrent visual states, and matched successful or failed recovery trajectories [166]. Reverse replay probes progress tracking without requiring a physically executable reverse trajectory. Its success-accuracy metric ranks successful against unsuccessful trajectories; it is neither the

robot's success rate nor calibrated acceptance at a deployment threshold. The results show that favorable progress correlation can coexist with weak instruction grounding or recovery assessment. Such offline evaluator evidence supports choosing diagnostic tests, but does not establish the closed-loop safety benefit of deploying that evaluator.

Evidence Limits The scope of governance evidence depends on both the enforced constraint and the test setting. ROSClaw exercises configured interception on TurtleBot3, with additional Go2 and G1 confirmation runs, whereas LabGuard tests its state-conditioned filter in simulation [156, 157]. ETA evaluates host-side checks in LIBERO and supplies qualitative UR5e recordings. The runtime-governance study tests synthetic policy violations in Gazebo UR5e simulation [155, 158]. The grasp loop and PhyAgentOS add limited physical validation of their decision paths [161, 56]. Formal assurance results further depend on modeled dynamics, state information, and timing. The resulting evidence is limited to the tested enforcement paths and the assumptions of each analysis.

The verifier itself is part of this evidence limit. The survey *No Free Checker* distinguishes the availability and cost of a check from the credibility of its verdict [167]. A check that arrives too late cannot support the same intervention as an immediate one, even if its eventual judgment is more accurate. Comparisons therefore need independent outcome evidence as well as the runtime verdict and its arrival time.

The next section examines the device interfaces through which these decisions take effect and execution evidence returns. Section 9 compares evaluation settings across the Harness responsibilities.

7 Digital–Physical Interaction in a Physical Agentic Harness

Digital–Physical Interaction is the Harness responsibility that connects an authorized operation to the physical interface that realizes it and returns evidence from that exchange. It preserves the binding, interface semantics, and spatial and temporal references needed to relate a command to its observations. The interface may be implemented through middleware, a service, or device-side software [168, 38]. Within these layers, Harness functions are identified by the binding and interpretation support they provide.

In this section, a *physical interface* is a runtime-visible interface through which the embodied system exchanges commands or observations with the physical task. It can expose robot sensors and actuators and, when the task depends on them, external instruments or environmental devices [169, 170]. The runtime resolves a logical capability invocation to a concrete endpoint and an interpretation, then dispatches the admitted operation and associates the returned status and observations with that invocation. Task Orchestration still selects the continuation, and Execution Governance still determines permission and outcome acceptance. Preparation can occur before admission, but a change in binding or interpretation requires the affected operation to be reassessed before dispatch.

We use *interaction session* for the record of one device exchange. It stores the selected endpoint, schemas, and reference configuration needed to interpret the exchange. This record differs from the objective-level execution session discussed in Section 3.2. Keeping the two records separate allows the runtime to distinguish a change in task progress from a change in the interface used to carry out one operation [56, 89]. Figure 7 summarizes these responsibilities.

7.1 Resolving Operations to Physical Interfaces

A capability request is logical until the runtime resolves it to an interface that can interpret and execute it. Discovery identifies candidate endpoints, compatibility checking examines their semantics, and binding selects the access path used for the exchange. EVA-Client separates robot backends, inference strategies, and transport middleware, allowing these parts of the interface to vary independently [171]. RIO uses replaceable real-time I/O components across robot configurations [172]. Such designs reduce platform-specific coupling, but a shared API remains valid only when the device assumptions behind it are preserved.

Middleware and device-side runtimes provide the transport, scheduling, and hardware abstraction on which an exchange depends. XBot2, for example, combines hardware abstraction with real-time and non-real-time components [173]. The ROS 2 real-time literature studies timing and scheduling at these lower layers [174].

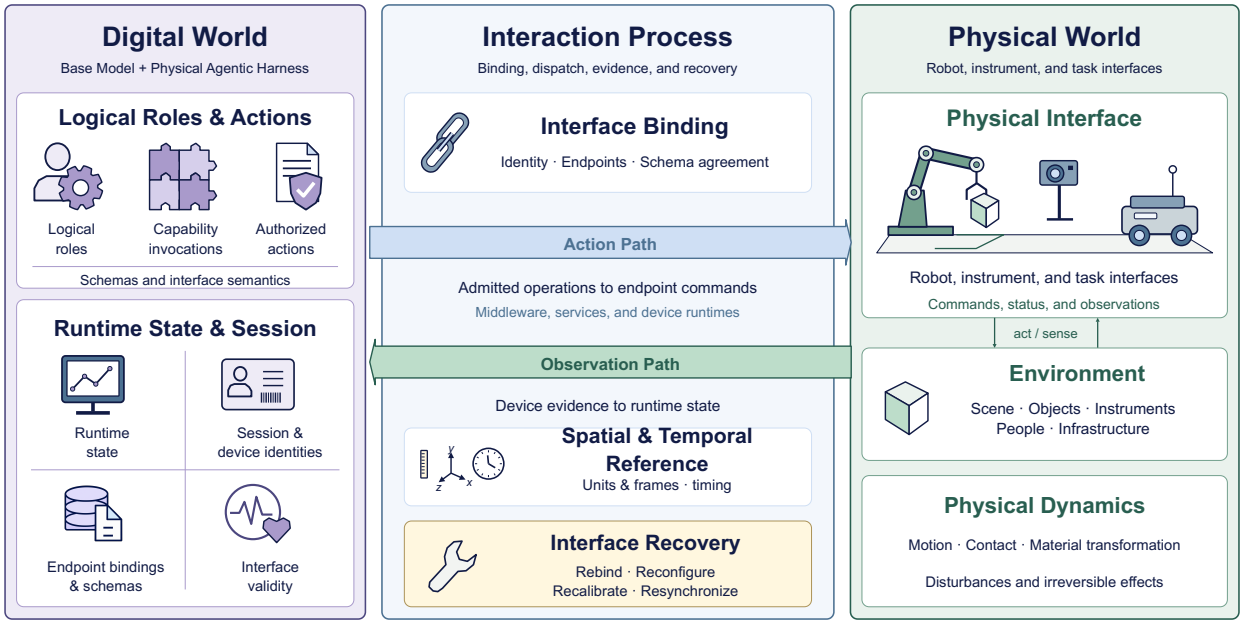


Figure 7 Digital–Physical Interaction as a device-grounded bidirectional process. The Harness runtime maintains logical roles, capability invocations, authorized actions, and interaction-session state. Device-side software and physical devices carry out authorized operations and return observations. Their placement does not by itself determine whether they perform a Harness function. Endpoint bindings and shared schemas connect the action and observation paths. Spatial and temporal alignment preserve their interpretation, while recovery restores the interaction through rebinding, recalibration, or resynchronization.

These mechanisms constrain when a request can be carried out and what feedback can be returned. They therefore supply execution support used by the Harness, while the Harness retains the binding and evidence needed for the task-level decision.

Discovering and Binding Devices Discovery identifies equipment and the capabilities it exposes. The candidate set may include robot sensors and actuators as well as instruments that affect the physical task. Identity, availability, and reachability answer different questions: a known endpoint may be busy, disconnected, or temporarily unavailable. Descriptions must also expose usable semantics, since structural identifiers alone do not explain how an operation should be interpreted [175].

Binding resolves a logical request to a device and endpoint, then records the configuration used for the exchange [176, 177]. A device replacement or endpoint change can invalidate that record even when the logical capability name stays the same. Trust and authority checks may restrict which endpoints or capability packages can be bound [178]. The final decision to permit a physical operation remains part of Execution Governance.

When several endpoints advertise the same operation, binding selects the access path and its interpretation rules. The device behind that path can reconnect or be replaced under the same endpoint name. Keeping the device identity separate from the access path lets the runtime distinguish these cases. Otherwise, observations from the replacement may be interpreted using assumptions retained from the previous device.

Agreeing on Schemas and Session State Schema compatibility concerns what a command or observation means. Types and argument names provide only part of this meaning; units, coordinate frames, controller conventions, and error semantics also affect interpretation [179, 180]. The same model checkpoint can produce different commanded motions under different action unnormalization and controller settings [181]. An interface check must therefore cover the path to the controller, including the adapter configuration. A unified API reduces repeated implementation work, but it does not establish this compatibility by itself [182, 183].

The interaction session retains the selected binding and interpretation rules during use. It also associates calibration and timing references with the exchange [56, 89]. Industrial telemetry shows why numeric values or local identifiers alone can be insufficient: reconstructing OPC UA data can lose engineering units and encounter identifiers that change across sessions [184]. Device identity and session identity therefore serve different purposes. Preserving both supports comparison across reconnection or configuration changes without assuming that a session-local identifier remains stable.

7.2 Dispatching Operations and Interpreting Evidence

Once an operation has been admitted, the action path translates it into the format expected by the selected endpoint. The feedback path returns execution status and measurements that can be interpreted under the same binding and schema. Perceptual capabilities may then turn measurements into task-relevant estimates; transporting a sensor value does not by itself establish the meaning or success of an action.

From Authorized Operations to Device Commands Dispatch translates the admitted operation into the command format expected by the selected endpoint. Adapters may convert parameter conventions or map a common API to platform-specific instructions [185, 179]. The execution unit may be a discrete call or an action stream. VLA-RAIL, for example, dispatches action chunks asynchronously [186]. A receipt or execution handle identifies the submitted work for later status queries. If rebinding or parameter conversion changes an assumption used during admission, the affected operation requires reassessment before dispatch.

Discrete calls and action streams expose different execution status. A discrete operation may return an identifier through which completion is queried. With a stream or sequence of chunks, some behavior may already have executed while later commands remain pending. The interface must report status at the execution unit it exposes. A received message establishes communication progress; it does not by itself establish how much physical behavior has occurred.

The uni-robot-agent harness makes observations, calibration, and robot state part of a consistent snapshot and uses request identifiers to deduplicate execution [187]. An operation with uncertain physical completion blocks further execution until recovery establishes its status.

Its real-robot evaluation uses task-rubric scores, which should not be reported as binary episode success rates.

From Device Feedback to Runtime State Device feedback describes execution status and observed conditions. Its interpretation depends on the source, acquisition time, and relation to the originating invocation [156, 171]. Normalization can expose a common observation format while retaining these references [188]. Execution traces provide further associations between calls and returned evidence [189, 190]. These records support task progress assessment and governance decisions. Missing or delayed feedback must remain explicit, since an absent update gives no assurance that the physical state stayed unchanged.

Normalization is useful only when it preserves the information needed by the consumer. Replacing acquisition time with arrival time hides the delay between sensing and processing. Associating feedback with the originating invocation also separates a late result from evidence about the currently active operation. Trace and session records preserve these associations for task-progress and outcome assessment.

7.3 Maintaining Spatial and Temporal Reference

Commands and observations remain meaningful only while their reference conventions apply. Spatial alignment relates a target or measurement to the relevant device. Temporal alignment establishes when information was acquired and whether it still supports the pending decision. Units, calibration, and controller conventions are part of the same interface state because each can change the physical meaning of an otherwise unchanged request.

Units and Spatial Alignment Unit conversion must preserve the intended physical quantity across the interface. Action normalization, sign conventions, and controller units can otherwise change the motion produced by an apparently unchanged request. Protocol-level unit types make some mismatches checkable before dispatch [180]. The active conversion settings therefore belong with the configuration used to interpret the operation.

Coordinate transforms relate sensor measurements, robot motion, and task targets. Transform infrastructures and joint spatial–temporal calibration provide established mechanisms for maintaining these relationships [51, 52]. More recent methods address asynchronous sensing and online estimation under changing conditions [191, 192]. For the Harness, the relevant question is whether the transform used by a command remains valid for the observation that supports it. A stored world frame does not supply the time-dependent pose of a moving sensor or establish that its calibration remains applicable.

Transform lookup and calibration address different parts of this problem. A system such as `tf` represents relationships between frames for downstream queries [51], whereas calibration estimates the parameters that make those relationships physically meaningful. Normal robot motion calls for an updated pose, not necessarily new calibration. A changed sensor mounting or reference configuration can instead invalidate the calibrated relationship even when transform lookup still succeeds. Know Your Body illustrates the complementary case in which queryable relations between commanded motion and functional points are revised from interaction feedback [193]. If such revisions are retained for later episodes, they enter the Persistent Evolution questions discussed in Section 8.3.

Timing and Observation Validity Interpretable timing requires known relationships between timestamps, including uncertainty about clock offsets, delays, and ordering [52, 194]. Clock offsets concern how timestamps relate, transport delay concerns when a measurement becomes available, and callback or execution order concerns when software processes it. Correcting one does not automatically correct the others. Deterministic ROS 2 execution addresses scheduling semantics, while temporal calibration addresses relationships between sensing streams [195].

Observation validity depends on the physical decision being supported. A recent reading can be incomplete, while an older observation may remain adequate for a slowly changing quantity. Inference can also introduce a gap between the observation used to decide and the state encountered during action [196, 197]. When freshness or alignment is uncertain, the interface should expose that uncertainty so the requesting component can obtain new evidence or reassess execution. Digital twins provide a setting in which observations update a maintained representation [198]. GaussTwin, for example, uses calibrated camera views to correct simulated object states [103]. Such state correction does not by itself change the calibration or the surrounding map.

7.4 Recovering from Interface Faults

An interface may become unusable even when the logical capability and task objective remain valid. The fault can concern connectivity, an adapter, calibration, or timing. Interface reconfiguration restores a usable exchange by producing a revised configuration and evidence that the configuration applies. This is a current-episode recovery mechanism. A revision retained for later episodes belongs to Persistent Evolution, discussed in the next section.

Detecting and Locating Interface Faults Receipts, endpoint status, observation timestamps, and session records help locate an interface fault [82, 89]. A missing receipt leaves delivery or acknowledgment uncertain; it does not establish that no physical action occurred. Additional status or physical observations may be needed before the attempt can be interpreted. Schema errors suggest an adapter problem, while inconsistent spatial or temporal references suggest calibration or timing faults. Perceptual misinterpretation and an invalid task plan require different responses and should not automatically trigger interface replacement.

Diagnosis can remain unresolved: a mismatch between the commanded target and returned observation may originate in the spatial mapping or in physical execution. Status and observation records narrow the investigation but may leave several plausible causes.

Repairing the Interface Rebinding changes the device or access path. Adapter reconfiguration repairs command interpretation, while recalibration and resynchronization address invalid reference relationships [192]. Some calibration procedures require physical motion, which remains subject to action admission. Restoring a connection does not justify repeating an earlier invocation whose physical outcome is unresolved.

Validating the Updated Interface Validation checks the part of the exchange affected by the repair: endpoint identity and schema after rebinding, command interpretation after adapter changes, or spatial and timestamp relationships after calibration and timing repairs. Restored network access alone does not establish that an interrupted physical action succeeded. The execution record should preserve the failed attempt and the configuration change so later assessment can distinguish their effects [56, 189].

The resulting evidence is passed back to the other Harness responsibilities. Execution Governance reassesses affected permissions, and Task Orchestration uses the observed state to select a continuation. Persistent Evolution concerns interface settings or other runtime support that are retained for later episodes.

8 Persistent Evolution in a Physical Agentic Harness

Persistent Evolution concerns revisions to Harness-managed components that remain available beyond the current episode. Retention means that a change persists; adoption means that a later episode uses it; improvement requires evidence of a benefit under stated conditions. A later episode may repeat the same task, so retained use and generalization to new task types require separate evidence. Temporary retries and reconnection remain current-run mechanisms. This section examines retained runtime support and separately managed components. The update target determines whether a change belongs to Harness evolution or Base Model training. A managed action policy can be revised as a capability, while another foundation model remains the Base Model. When the action policy is itself the Base Model, changing its core weights is a model update; changing its surrounding memory or execution support is a Harness update. Joint approaches can involve both.

8.1 Targets of Persistent Evolution

A retained revision can affect any of the other five responsibilities. The following targets identify what changes, without requiring a separate module for each responsibility.

Task Orchestration Saved workflows can change step order, skill routing, or handoff and recovery conditions. Replanning only the current task is distinct from retaining a revised procedure for later use.

Digital Resources Experience can revise reusable memory, predictive models, or retrieval rules. For example, a recurring failure can motivate a retrieval rule used in later episodes; merely saving its observation does not establish such a revision.

Embodied Capabilities A library can acquire a new skill or replace an implementation, policy, or adapter. Changes to the required inputs or action conventions can also affect callers and downstream capabilities.

Execution Governance Stored constraints, monitoring rules, and acceptance criteria can change. An intervention affects current execution; a retained rule changes how future operations are assessed.

Digital–Physical Interaction Experience can motivate a revised binding, calibration procedure, or command interpretation that is retained for later episodes. Routine state refresh and reconnection during the current exchange do not establish this change.

8.2 Evolution Process

Figure 8 links four activities from execution feedback to later use. Development tests help construct a candidate, separate evaluation tests its behavior beyond the development examples, and release controls determine where it becomes active. The ordering and extent of these activities depend on the component and its risks.

Collecting and Interpreting Feedback Feedback becomes useful for revision when an observed outcome can be related to the component that may have caused it. A failed grasp can reflect perception, execution, or unsuitable task guidance, so a failure label alone does not select the update target. Thea uses evaluator-approved outcomes to consolidate lessons, while ENPIRE gives coding agents rollout and verification results for repeated physical experiments [89, 199].

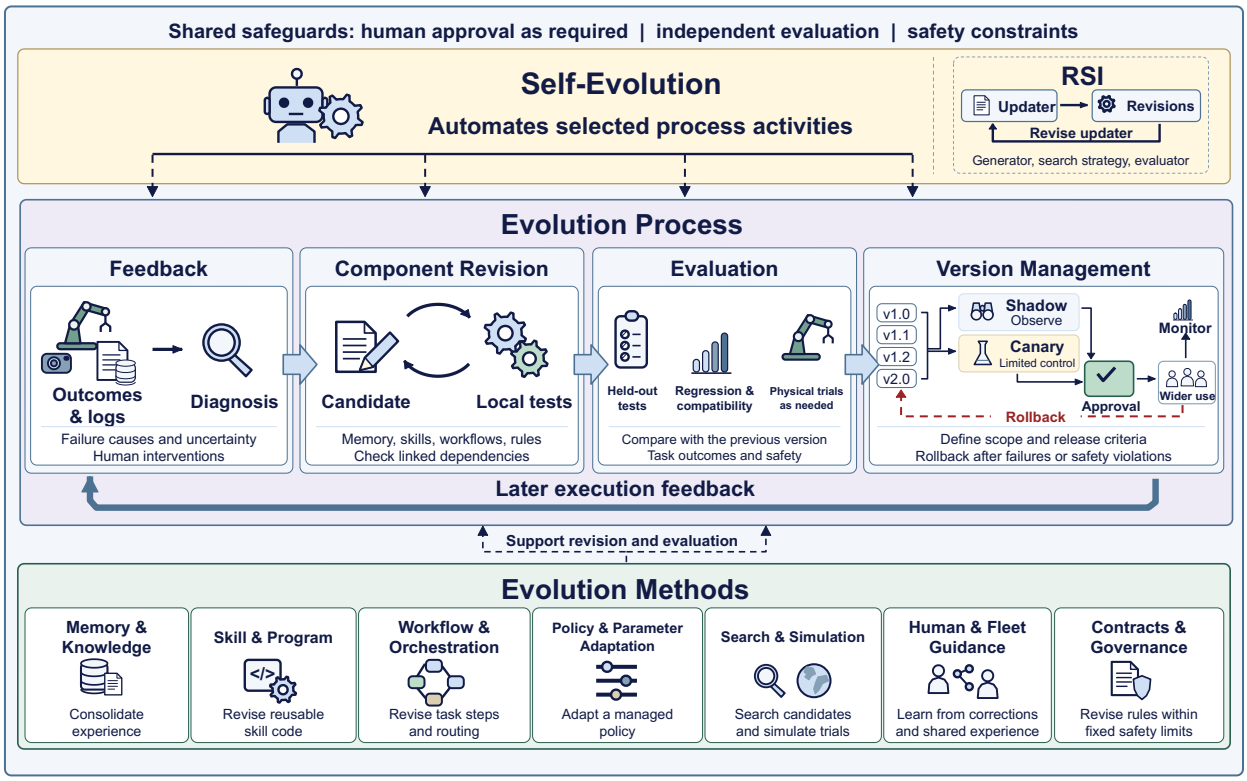


Figure 8 Three complementary views of Persistent Evolution. The top panel shows how Self-Evolution automates selected process activities, with recursive self-improvement (RSI) also revising the procedure that produces future updates. The middle panel connects feedback, component revision, evaluation, and version management, with later execution providing feedback for further revisions. The bottom panel summarizes seven groups of methods that support revision and evaluation. The outer frame represents shared safeguards across all three panels, including human approval when required, independent evaluation, and safety constraints. Individual systems may implement a subset of these functions and select deployment controls according to their risks.

Zeva clears its online memory between episodes; Remember Smarter and AGM also use execution history to support current-task progress [200, 201, 202]. They illustrate episode-local support, whose persistence would need to be established separately.

Revising Harness Components Revisions may need to change several support components together. Teach and Grow links reusable skill blocks with structured experience memory in LIBERO [203]. A skill update that expects a new memory representation requires a compatibility check with the reader that supplies it. The candidate evaluation therefore covers the change and its affected dependencies, whether generated manually or automatically.

Evaluating the Revised Harness Update evaluation asks whether a candidate achieves its intended change and whether previously supported behavior regresses. The intended change may repair a failure, add a reusable skill, or revise how evidence is retrieved. Tests of that objective need to be accompanied by regression and compatibility checks, including tests beyond the development examples. EmbodiedSkills supplies typed execution checks and recovery traces in simulation, whereas REFACTOR-VLA evaluates library abstractions on offline LIBERO data [65, 204]. The latter's reward-based check is inactive when reward annotations are unavailable. These studies provide evidence from simulated execution checks and recovery traces, and from offline library evaluation, respectively. Neither establishes subsequent physical adoption of a revision.

The evaluation unit also depends on what the revision changes. Replacing a skill implementation can affect the states supplied to its successors even if the replacement succeeds in isolation. A revised memory reader can change the evidence supplied to an otherwise unchanged policy. Testing only the edited component would miss these interactions. Update evaluation therefore includes dependency compatibility, with performance claims limited to the tasks tested.

Managing Harness Versions Version management connects an adopted component to the evidence used to select it. Qin et al. evaluate gated promotion and rollback for versioned skill modules in robosuite [205]. Their gate blocks injected regressions but also rejects some clean candidates, while the rollback detector misses some deployed regressions. The observed false rejections and missed regressions limit the reliability of these release controls. ICAN-Deploy separately examines capability identity and deployment mechanics in bridge tests [206]. AgenticRobotics links versioned evidence and approval in simulated LIBERO-10 campaigns [207]. Because a capability name can remain stable across changes, tracking the actual implementation and its dependencies is necessary to interpret later behavior.

Systems can use deployment controls to limit exposure to a candidate; the reviewed implementations vary in which controls they provide. *Shadow deployment* compares candidate decisions on live inputs without granting it robot control. *Canary deployment* limits candidate-controlled execution to selected tasks or robots, and *promotion* expands that scope after review. Shadow comparisons leave the physical consequences of candidate actions untested, whereas canary operation incurs those consequences within its permitted scope.

Rollback restores a compatible configuration for future execution; it cannot undo the physical effects of earlier actions. A restored skill may also require a compatible memory reader or adapter. Subsequent failures must therefore be interpreted against both the active version and the physical state left by prior execution. The reviewed version-management studies provide limited evidence of this complete process over sustained physical operation.

8.3 Evolution Methods

Revision methods can be compared by the object they change and by the support used to generate or validate that change. We first examine methods for memory, skills, workflows, and managed policies. We then discuss search, human and fleet feedback, and governance checks that can support several revision targets. A method can combine several of these update targets and supporting mechanisms.

Memory and Knowledge Evolution Memory evolution revises guidance that can be reused in later episodes. Thea consolidates successful task lessons into durable memory and tool-specific experience used in later sessions [89]. Its sessions can each contain several tasks. PhyAgentOS checks knowledge before cross-session reuse, but a child session may continue an unfinished goal [56]. Such continuation remains within the same episode under our definition in Section 2.2. Cross-episode adoption requires use after the earlier episode ends. These systems describe physical implementations of retention and reuse, but their task demonstrations do not isolate the cumulative benefit of every retained lesson.

Know Your Body retains a different kind of knowledge: relations used to interpret robot motion and reusable experience derived from interaction [193]. Revising a body estimate can require checking experience that depended on the old estimate before reusing it. This links persistent knowledge revision to the spatial interpretation discussed in Section 7.3. Its physical experiments include progress across successive successful trials, but those curves are conditioned on success and do not establish an unconditional improvement in success rate.

Controlled studies examine changes to reusable content and to the mechanisms that select it. OnEvoMemory updates its value estimator and memory-selection modules around a frozen policy; episode-local memory is distinct from these learned revisions, evaluated after a round of adaptation on manipulation benchmarks [208]. ABot-AgentOS instead turns diagnosed retrieval and answering failures into structured assets that affect only later data splits, with evolution evaluated on embodied question-answering and question-answering benchmarks [209]. Harness VLA combines task-specific reference traces with general rules derived from failure analysis; its few-shot LIBERO and RoboCasa evaluations test this guidance on held-out seeds [26].

DreamLedger retains execution histories used to decide whether to rely on world-model predictions, evaluated in simulation and Franka tabletop experiments [210].

The revised object determines how reuse can be evaluated. A task-specific trace can be checked on new executions of the same task, whereas a general rule calls for tests beyond the case from which it was derived. For a revised selection mechanism, the question is whether it retrieves more applicable evidence, even if the stored records are unchanged. Histories used to assess predictions instead need to be tested through the resulting decisions to trust or reject those predictions. Evaluation must therefore measure how the revised content or selection mechanism changes later decisions.

SERA supplies a controlled example of retained guidance: its selected harness improves RoboMME success from 44.38% to 51.38% across 800 test episodes, with planning and action models fixed [39]. The test uses new instances of the same task set, and development can use privileged diagnostic traces that are unavailable at runtime. InstAgent compares initial and evolved experience libraries after freezing both for physical evaluation [40]. Its three model-specific comparisons improve from 34/40 to 38/40, 31/40 to 38/40, and 28/40 to 34/40 successes; the paper's significance tests against its no-memory condition should not be reassigned to these initial-versus-evolved contrasts. AMP1 separately studies task-specific practice and memory accumulated across a task sequence [211]. The scope of its reuse evidence depends on best-of-five practice selection in one protocol and a single continual-task ordering in the other.

Skill and Program Evolution Skill revision can change either an executable implementation or the guidance used to invoke it. Lifelong Robot Library Learning builds composable library entries through demonstrations and exploration [212]. EmbodiSkill distinguishes defects in textual guidance from execution lapses before editing a skill, whereas PRACTICE maintains skill cards through explicit library operations [154, 213]. Editing skill guidance leaves the underlying controller unchanged.

Code-level methods instead revise the operation itself. ASPIRE repairs code-as-policy candidates and retains repair patterns, with some real-robot evaluation; ReSYNC derives reusable concepts and skills from recovery [153, 214].

Skill1 and SkillRL supply related learning-based library mechanisms in digital settings [215, 216].

Exploration can also produce reusable support before a downstream task is specified. Playful Agentic Robot Learning builds a library of executable skills through agent-directed play, then freezes it for downstream evaluation [217]. World Action Agent instead converts video or human teaching into multimodal skill guidance through an edit-and-review process [133]. Its library is constructed on source tasks and frozen before testing on other simulated task settings. Their downstream tests measure reuse of fixed code or guidance libraries; the libraries remain unchanged during testing.

Recursive Harness Distillation uses a stronger teacher to revise guidance for a smaller agent around a frozen action policy [218]. Its physical evaluation improves from 28/75 successes for the action policy to 48/75 with the refined playbook, after task-specific policy fine-tuning and separate physical-playbook adaptation. This comparison does not test zero-shot transfer of the simulation playbook. RoboArchitect searches executable harness code and persistent memory, allowing the resulting agent to bypass the VLA through direct control [219]. Its aggregate combines benchmark-specific success and progress measures, so that aggregate should not be labeled a single success rate. HarnessPAI also retains executable support and failure–repair experience [220]. Its evaluation protocols must be distinguished: reported LIBERO-family seeds include development seeds, whereas its robosuite protocol separates development and evaluation seeds. SimEX freezes an adapted toolbox for subsequent tests after simulation search and a limited hardware-adaptation phase [221]. Interpreting these results requires identifying the retained artifact and the tests kept separate from its development.

Workflow and Orchestration Evolution Workflow evolution changes how capabilities are connected across tasks. GaP searches workflow graphs in simulation before physical demonstrations, while Don't Drop the BATON retains information about skill handoffs in simulation [222, 75]. Graph search changes the procedure; retained handoff information changes how a transition is prepared. RoboHarness links policy-specific experience to orchestration, and Embodied Lifelong Learning accumulates planning experience in simulation [116, 223]. HyMeS couples executable-memory updates with constraint-rule revisions around a

frozen policy, evaluated in simulation and limited SO-101 trials [224]. These approaches expose different dependencies between the revised workflow and the skills it calls.

A workflow revision can give an unchanged skill a more suitable starting state by altering call order, branching, or handoff context. Later gains may therefore arise from how the skill is used, with its learned behavior unchanged.

Policy and Parameter Adaptation Learning methods can revise action implementations that a Harness manages. To study such a revision as Persistent Evolution, the managed component and its retention across episodes must be identified; claims about later use additionally require adoption evidence. Self-Improving Robots updates a visuomotor policy through physical practice [225]. We include it here as a method for revising an action implementation. RISE instead uses imagined rollouts from a compositional world model for policy updates and reports gains on three real-world manipulation tasks [226]. It supplies a different source of update experience. For both studies, the cited evidence concerns policy improvement under the tested conditions. Their role in a Harness depends on which component is managed and whether the revised implementation is retained for later episodes; a benefit from reuse must be demonstrated in those later episodes.

Q-Planning separates a frozen action-proposal policy from an updated value function [227]. The value function weights candidate action chunks to form the executed action. Behavior can therefore change while the proposal policy's weights remain fixed. If an updated policy is itself the system's Base Model, its weight change remains model adaptation under our definition. Its surrounding evaluation and adoption process can still provide Harness support, but the two contributions require separate evidence.

Robo-COP couples policy updates with memory revision and checkpoint-dependent reconciliation [49]. It evaluates frozen configurations after deployment-time learning, with controls for frozen policies and fixed schedules. Its physical comparison improves mean success from 38.3% to 50.0% over three tasks, with the gain concentrated in one task; the joint result does not isolate memory revision alone. BioAgent similarly couples skill acquisition with reusable protocol support, while Self-Building Labs constructs digital-twin and device support under human preparation and oversight [228, 229]. Their physical demonstrations substantiate these specific update targets.

The following approaches support several revision targets through candidate search, human and fleet feedback, or checks on adoption.

Search and Simulation for Evolution Search explores alternative prompts, code, workflow structures, or parameter settings before selecting candidates for further testing. SHAPER optimizes textual skills and executable supporting code while keeping its planner and executor frozen in simulation [36]. Zetta combines failure clustering, replay, regression and held-out tests with versioned switching in simulated benchmarks [37]. DINO-WM and Habitat provide predictive or interactive environments that can support candidate screening [95, 97]. They provide testing resources; demonstrating later use of a selected revision additionally requires adoption evidence. Search can exploit simulator errors or evaluator shortcuts, so screening results require checks under the conditions relevant to physical use.

Human- and Fleet-Guided Evolution Demonstrations, corrections, takeovers, and shared robot experience help select and produce revisions. Robot Learning on the Job and Fleet-DAgger turn human assistance into later policy updates [230, 231]. Multi-Task Interactive Robot Fleet Learning studies assistance across tasks, while AutoRT coordinates tasks and supervision [232, 25]. Learning while Deploying reports shared replay and deployment benefits across sixteen AGIBOT G1 robots [233]. Assistance allocation and data collection supply experience for updates, whose adoption must be tracked separately. Reusing experience across robots also requires checking the configurations and task conditions under which it was collected. An apparent fleet-wide benefit must therefore be distinguished from improvement on the robots and tasks contributing most of the data. Providing training evidence and authorizing release are separate roles, as discussed in Section 8.4.

Contract- and Governance-Guided Evolution Contract-based methods make candidate compatibility and adoption conditions explicit. ECM Contracts formalizes versioned interfaces, and Governed Capability Evolution adds sandboxing, gated activation, monitoring, and rollback [148, 234]. Harnessing Embodied Agents connects these concerns to policy-constrained runtime execution [158]. HarnessX isolates harness variants in digital-agent settings [235]. EmbodiedGovBench examines governance and upgrade safety [236]. Their contribution is to constrain which revisions may be adopted and under what conditions, with physical safety requiring evaluation in the intended setting.

8.4 Self-Evolution

Self-Evolution automates activities that produce, test, and retain revisions for later operation. Its scope is the automation of the update process across the targets described above. Automation can cover only part of that process, and human approval can remain necessary before physical use.

Automation Scope Self-Harness automates revisions to the runtime around a fixed model [22]. Execution traces supply recurring failure patterns, and the same model proposes bounded harness edits to address them. A fixed evaluator tests candidates before adoption. Its experiments on Terminal-Bench-2.0, SWE-bench Verified, and AppWorld concern digital execution support. The held-out split participates in the adoption gate, so it is not an independent final test.

ENPIRE instead uses coding agents to edit policy and training code through repeated real-robot trials [199]. Its relevance here is the automated experiment process that tests successive candidates and feeds physical results back to the coding agent. The policy changes and the support that organizes their evaluation are distinct contributions. The demonstrated automation is confined to this update loop within the prepared task environment.

Harness Continual Learning studies revisions to interfaces, memory, capabilities, and routing in non-physical experiments [237]. It supplies a methodological comparison for updates to runtime support, with a different validation setting. A retained change may also remain unused, as the distinction between updating and benefit in Harness Benefit makes explicit [238]. An automated revision therefore needs evidence of later adoption before it can explain later behavior.

Several recent systems automate narrower update loops. DynaHarness retains revised execution support but relies strongly on its analytic capability library; FastHarness retains experience for scheduling supervision [73, 163]. CarryOn's later recovery rounds accumulate additional successful recoveries from the original failed cases [159]. Those cumulative results include renewed attempts and are not independent success estimates for a frozen updated harness. SEAR explicitly compares reusable libraries and budgeted development, including cases where incorrect verifiers propagate through retained experience [48].

Human Oversight Human participation depends on the stage. A person may supply corrective experience, approve a physical trial, or authorize wider use of a candidate. Supervision during operation is a different role from approval of the update. These arrangements can coexist with automated candidate generation and testing. Their scope determines which parts of a reported self-evolution loop actually operate without human decisions.

Recursive Self-Improvement Recursive Self-Improvement (RSI) revises the procedure that produces future updates, such as its search strategy or evaluator. Repeated component revisions under an unchanged updater do not establish this additional capability. HiSME studies joint evolution of task skills and a meta-skill in BFCL-v3 and MineDojo, while Harness-R1 trains an updater using patch outcomes in WebShop, ALFWorld, and DBBench [239, 240]. These methodological examples remain to be validated on physical systems. RegenHarness proposes an evidence-gated revision process with regression checks and release comparisons for a physical robot runtime [165]. Its hardware cases exercise execution and evidence handling, but the paper does not report controlled longitudinal results for recursive self-improvement. For physical systems, evaluating an updater revision requires observing its effects on subsequent update cycles.

Flex-RSI makes the recursive distinction operational by allowing the improver's instructions and update artifacts to change alongside the executor [241]. Its mixed digital and simulated-robotics experiments provide evidence for that editable architecture. However, enabling the entire improvement process does not isolate

the causal benefit of modifying the improver, and validation gains are distinct from final hidden-test gains. InstAgent, Recursive Harness Distillation, and RobotAgentBench also use recursive terminology, but the evidence discussed here primarily concerns repeated program, tool, or guidance revision [40, 218, 242]. Under this survey's stricter definition, repeated artifact improvement alone is insufficient evidence that a revised updater improves subsequent update cycles.

9 Evaluation and Applications of Physical Agentic Harnesses

Evaluation provides evidence about how the six Harness responsibilities support physical tasks. Task benchmarks measure behavior under specified conditions, while application studies show how runtime mechanisms work together in a particular setting. Both can involve simulation or hardware; neither label alone establishes deployment maturity. The bottom panel of Figure 2 highlights representative benchmarks and applications examined in this section. We distinguish benchmarks of task competence from tests that expose coordination, recovery, or governance decisions, then examine applications on physical devices.

9.1 Task and Capability Benchmarks

Task benchmarks provide controlled settings for comparing the behavior of an embodied system. LIBERO varies spatial arrangements, objects, goals, and task sequences to study knowledge transfer and lifelong robot learning [243]. RoboCasa broadens simulated manipulation to kitchen activities, including composite tasks. RoboCasa365 extends this setting with more tasks and environments for studying generalization [244, 245]. These environments support comparisons of capability selection or execution support when the underlying policy is held fixed. LIBERO's transfer and retention measures address learning across task sequences, whereas kitchen benchmarks emphasize success across task and environment variations.

Bimanual manipulation adds dependencies between the motions of two arms. RoboTwin tests these dependencies through object handovers and tasks where the arms must avoid mutual interference [246]. RoboTwin 2.0 broadens evaluation to 50 simulated tasks and compares policy success under clean and domain-randomized conditions [247]. These scores assess bimanual task performance and robustness, but do not isolate the contribution of Harness coordination.

Long-horizon activities expose dependencies that short manipulation trials may omit. BEHAVIOR-1K defines everyday activities in OmniGibson, whose simulation includes rigid and deformable objects and liquids [248]. Such tasks require intermediate physical conditions to remain suitable for later steps. EmbodiedBench instead compares vision-driven agents across environments with different planning and action demands [249]. The former supplies detailed activity and physics settings; the latter helps compare how multimodal models use observations and available actions. Comparisons between models need to hold the observation and action interface fixed, because that interface determines the information and operations available to each model. When interface design is itself under study, it becomes the comparison variable; the underlying task, model, and other execution conditions need to be controlled or their differences reported.

CaP-X makes this interface dependence an explicit evaluation variable [108]. Its CaPBench compares coding agents across action abstractions, feedback forms, and single- or multi-turn execution. The resulting task-success comparisons show that additional feedback is not uniformly useful: its form and the available primitives affect whether the agent can turn it into a correction. This setting complements task-diversity benchmarks by examining how a fixed agent's behavior changes with its execution interface.

Hardware benchmarks address a different source of uncertainty: whether an evaluation setup captures the conditions encountered by a physical robot. RADAR measures task success and translational error under changes in objects, robot initialization, sensing, and instructions [250]. VLA-REPLICA provides a low-cost setup and tests policies under in-distribution and out-of-distribution conditions [251]. These studies complement simulation by testing physical variability and repeatability. RADAR stresses variation within an instrumented setup; VLA-REPLICA tests reproduction on a separately constructed SO-101 setup, using a subset of tasks with high original success rates.

Reproducibility also depends on the software implementing an evaluation. vla-eval separates model serving from isolated benchmark environments and records the configuration used for each run [252]. Its reproduction study traces discrepancies to preprocessing and action-interpretation choices that can otherwise be mistaken for model differences. The reported framework integrates fourteen simulation benchmarks, with cross-codebase reproduction checks on three. This evaluation infrastructure makes the experimental interface inspectable and repeatable; physical deployment lies outside the reported study.

Recent benchmarks expose additional parts of the agent–robot interface. CodeActionBench hides verifier outcomes while allowing direct calls and generated programs through the same robot API; repeated attempts use one fixed scene per task [253]. RobotWorld broadens embodiment and task coverage, but its reported comparison has one episode per model–task pair and pauses physics during model reasoning [254]. RoboQuest tests information-seeking manipulation with a physical submission action, distinguishing missing evidence from an action that was attempted and failed [255]. Embodied Playground combines goal, rule, and physical-validity checks; its analysis also identifies incomplete disclosure of rules or thresholds in many failed episodes [256]. These scores therefore require interpretation against the information actually supplied to the agent.

EmbodiedSkillBench compares skill provision under a common interface and shows that additional skills need not help every model [257]. On its fixed Clean100 setting, replacing native control with skill-only execution raises GPT-5.6-sol success from 11% to 63%, but reduces GPT-6-astra success from 92% to 61%, while reducing execution cost in both cases. When GPT-6-astra retains native control and can select skills through Agent-Directed Reuse, success reaches 85%; the restriction on available control therefore matters to the comparison. These are model-specific comparisons on one evaluated seed, not evidence that either richer libraries or direct control is uniformly preferable. RoboDojo complements simulation with standardized physical manipulation trials and explicitly separates binary success from partial-progress score [258]. Its separate simulation and hardware suites cover complementary tasks, leaving direct transfer untested. RoboColosseum adds distributed physical comparisons across sites and embodiments; paired open-track comparisons and the separate fine-tuning track answer different questions [259]. Neither a shared task nor shared demonstration data alone ensures identical model-training or inference conditions.

9.2 Benchmarks for Coordination, Recovery, and Governance

Coordination benchmarks differ in how they separate task completion from cooperative behavior. PARTNR evaluates simulated household collaboration with spatial and temporal constraints and agents with different capabilities [260]. It reports success and percent completion of task propositions; simulation steps and task offloading further reveal whether cooperation reduces work. Its comparisons vary planning, perception, and skill execution, making the source of an improvement dependent on the configuration being compared.

CoCoBench fixes a high-level skill executor and scores allocation, precedence, mutual exclusion, and handoff behavior separately [47]. Its construct scores measure reference-relative execution time, ordering violations, resource contention, and buffer handoff behavior. In the centralized image-observation protocol, Claude Haiku 4.5 completes 89.4% of the 218 allocation instances but obtains a coordination score of 0.48, showing that completion can coexist with inefficient allocation. These episodes use task-specific, oracle-calibrated skill-step budgets, with budget exhaustion counted as failure; the construct score assesses the execution trace rather than another success rate. This diagnostic separation complements PARTNR's broader task-level collaboration analysis.

Recovery evaluation changes the initial condition instead of only the task instruction. LIBERO-RECOVER begins with states left by failed manipulation attempts and measures whether the original task can subsequently be completed [85]. Its retry, action-adaptation, object-state, and environmental-recovery categories distinguish how much of the execution context must change. The resulting score concerns recovery from those simulated failure states, whereas ordinary nominal-initialization trials combine failure avoidance and recovery. RoboRecover extends same-state comparison across RoboTwin and LIBERO by replaying action prefixes to reconstruct execution deviations [261]. Its scenarios combine natural policy failures with action-perturbed and human-constructed cases. It reports original-task recovery success overall and by task stage and deviation type, exposing differences hidden by nominal-start success. Replay gives compared policies

a matched recovery point, while the sampled failure distribution and renewed interaction budget define the scope of the score.

REBOOT complements these simulated recovery settings with phase-annotated physical assembly data [262]. It provides nominal demonstrations and expert recovery trajectories on a bimanual WidowX AI platform. Its policy evaluation measures task success and where execution fails, distinguishing alignment and engagement errors that a binary outcome merges. The released recovery data supports subsequent policy development, but the paper leaves the choice of how to train on those corrective segments open. Its contribution is physical failure analysis and data for recovery supervision; a general recovery controller remains to be validated.

Safety benchmarks separate instruction completion from acceptable behavior. SafeAgentBench varies explicit, implicit, and long-horizon hazards, measuring rejection, execution, and success under goal-based or language-model judgments [263]. For hazardous instructions, a high completion rate can indicate unsafe compliance. Its long-horizon analysis further distinguishes completed-and-safe plans from completed-but-unsafe plans, so the relevant outcome depends on the instruction category and evaluator.

EmbodiedGovBench tests a different boundary: whether runtime decisions respect authorization and recovery conditions under perturbations [236]. Its measures include unauthorized invocation, recovery containment, drift detection, and upgrade-related permission violations. The reported instantiation covers part of the proposed framework and uses simplified comparison systems; its aggregate governance weights are not empirically calibrated. These limitations matter when interpreting its scores as evidence about a specific enforcement mechanism.

RobotAgentBench compares planner, executor, and verifier combinations in randomized household scenarios and includes a matched simulation–hardware subset [242]. Its scenario progress and recovery measures do not reduce to whole-task success. Its iterative tool-refinement setting also differs from a frozen, independently tested final harness. SEAR evaluates retained skills and programs under explicit development budgets and separate tests, while reporting both simulation and a smaller physical suite [48]. Its findings include propagation of incorrect self-written verifiers, motivating outcome checks independent of the evolving agent. Flex-RSI supplies a related evaluation setting in which both executor and improver artifacts may change, but its mixed digital and simulated-robotics results are not evidence of physical deployment [241].

Table 1 compares the settings and measurements. Their scores answer different runtime questions and cannot be ranked on a common performance scale.

Table 1 Representative benchmarks and evidence boundaries. Comparisons distinguish information access, adaptation and test separation, and physical validation where reported. Success, progress, recovery, and evaluator scores retain their protocol-specific meanings and are not ranked across benchmarks.

Benchmark	Setting and target	Comparison and protocol	Main measures	Evidence limit
LIBERO [243]	Simulated manipulation and transfer	Spatial, object, goal, and task-sequence changes	Forward / negative backward transfer; success-curve AUC	Transfer scores depend on task order and learning protocol
RoboCasa / RoboCasa365 [244, 245]	Simulated kitchen manipulation	Task and environment diversity	Task success across test conditions	Success depends on task split and environment variation
RoboTwin / RoboTwin 2.0 [246, 247]	Simulated bimanual manipulation	Task and object variation; clean vs. randomized scenes in 2.0	Task success by task and test condition	Depends on policy and training data; does not isolate Harness contributions
BEHAVIOR-1K [248]	Long-horizon simulated activities	Tasks with varied objects and physical state changes	Task success and goal completion	Goal predicates and modeled state changes define completion

Table 1. Cont.

Benchmark	Setting and target	Comparison and protocol	Main measures	Evidence limit
EmbodiedBench [249]	Vision-driven agents in simulated environments	Models, tasks, and action demands	Task success across environments	Observation and action interfaces differ across environments
CaPBench [108]	Coding agents for simulated manipulation	Action abstraction; feedback; single- or multi-turn use	Task success by interface setting	Results depend on available primitives and feedback access
RADAR [250]	Real-robot VLA generalization	Physical and sensing changes; spatial tasks	Task success; translational error	Instrumented outcomes cover the selected spatial tasks
VLA-REPLICA [251]	Reproducible real-robot manipulation	Distribution shifts; independently built setups	Task success under in- and out-of-distribution conditions	Reproduction tests a selected high-performing task subset
PARTNR [260]	Household collaboration in Habitat	Planning, perception, skills, and partner configuration	Success; percent completion; steps; task offloading	Completion metrics are agent-agnostic; efficiency needs separate measures
CoCoBench [47]	Team planning in AI2-THOR	Team size; centralized or local planning; communication	Success; four trace-based coordination scores	Fixed skill executor; scores require construct-relevant events
LIBERO-RECOVER [85]	Simulated manipulation after failure	Failure states requiring different recovery scopes	Original-task completion after recovery	Scores depend on the sampled failure states and recovery budget
RoboRecover [261]	Recovery in RoboTwin and LIBERO	Replayed deviations; task stage; deviation type	Overall and stage–type recovery success	Filtered failure distribution; fresh budget after replay
REBOOT [262]	Physical precision assembly	Assembly / disassembly; policy; failure phase	Task success; phase-level failure counts	Controlled setup; recovery-training design remains open
SafeAgentBench [263]	Hazardous instructions in simulation	Explicit and implicit task hazards	Rejection; goal/LLM success; completed-and-safe plans	Metric interpretation differs for safe and hazardous instructions
EmbodiedGov-Bench [236]	Governance perturbations in simulation	Authorization, drift, recovery, upgrades, and override	Unauthorized calls; drift detection; recovery containment	Partial scope; simplified baselines; uncalibrated score weights
RoboDojo [258]	42 simulated and 18 hardware manipulation tasks	Separate sim / hardware suites; repeated standardized physical trials	Binary success and partial-progress score; simulation aggregates by capability dimension	Not a paired sim-to-real transfer test; human reset and physical scoring

Table 1. Cont.

Benchmark	Setting and target	Comparison and protocol	Main measures	Evidence limit
CodeAction-Bench [253]	25 simulated dual-arm tasks	Hidden verifier; common API and budgets; three attempts on each fixed scene	Attempt success; task coverage; completion-claim accuracy; cost	One scene per task; vendor-harness comparisons may also change the model
RobotWorld [254]	84 simulated tasks across embodiments	Task-specific sensing / control contracts; one episode per model–task pair	Binary task success; interactions and wall time	Physics pauses during reasoning; no repeated-trial uncertainty estimate
RoboQuest [255]	Information-seeking mobile manipulation in simulation	Ten tasks; 50 instances each; hidden task information and physical submission	Submitted-task success; partial progress; attributed failure types	Information acquisition and motor execution differ; isolated-skill success is not full-task success
Embodied Playground [256]	Long-horizon simulated agentic tasks	EP100 under four conditions per task; reported runs use L3 privileged observations	Pass@1 / Pass@2; goal, rule, and physical-validity checks	Incomplete disclosure of rules or thresholds affects failure interpretation
EmbodiedSkill-Bench [257]	Skill use on 100 simulated LIBERO-Pro cases	Common interface; native control, skill-only execution, retrieval, or agent-directed reuse	Task success; accepted action calls; tokens and elapsed time	Single evaluated seed; skill benefit depends on the model and execution restrictions
MeMoREval [94]	53 memory-dependent tasks in four simulated worlds	Shared prior experience; no new observation during requests	Task, question, and action success; memory write / read cost	Simulator-derived spatial records; action plans execute without feedback repair
RoboValue [166]	Offline value judgments on simulated and real trajectories	Separate zero- / one-shot tracks; environment and embodiment shifts	Outcome ranking; instruction grounding; progress, failure, and recovery judgments	Value-model metrics are not robot success or closed-loop safety guarantees
RobotAgent-Bench [242]	Household simulation and matched hardware subset	Planner / executor / verifier combinations; separate iterative tool refinement	Task scores; scenario progress; recovery and stagnation	Different score meanings; iterative refinement is distinct from frozen final-test evaluation
RoboColosseum [259]	Distributed physical evaluation across sites and embodiments	Paired open-track comparisons; separate demonstration-based fine-tuning track	Difficulty-calibrated ranking; task success and progress	Coverage differs across systems; shared data does not equal matched training compute

Table 1. Cont.

Benchmark	Setting and target	Comparison and protocol	Main measures	Evidence limit
SEAR / Flex-RSI [48, 241]	Retained support in SEAR; editable executor and improver in Flex-RSI	Development budgets and separate tests; SEAR also includes a small hardware suite	Post-development task-specific scores; reuse and update comparisons	Mixed metrics and scopes; Flex-RSI is digital / simulated, not a hardware RSI demonstration

9.3 Applications in Physical Environments

Physical applications expose dependencies among mechanisms that benchmarks often study separately. Even a competent skill can receive an unsuitable target or leave an outcome that the runtime misjudges, causing later operations to start under obsolete assumptions. The studies below illustrate these dependencies in different environments. Their evidence ranges from controlled hardware trials to field operation, with different roles for human preparation and supervision.

Household Manipulation Household tasks require perception and action to agree on objects that were not specified in advance. OK-Robot combines open-vocabulary object localization, navigation, and grasping on a mobile manipulator, with evaluation in real homes [128]. Its integration shows why individually capable models still depend on consistent representations and reachable targets. The system relies on an initial environment scan, and its static map limits adaptation to subsequent scene changes.

Thea makes outcome evaluation part of tool execution, connecting scene-graph context to retry and continuation decisions [89]. Its main comparison uses Astribot S1 and holds the decision model and underlying tools fixed across Thea, CaP-X, and SayCan [108, 31]. Across single-object, multi-object, and navigation-plus-manipulation tasks, the reported advantage grows with task complexity; the study uses 20 trials per method at each of the first two levels and 15 at the third. The unspecified common retry cap limits this result to a qualitative architecture comparison; it cannot establish single-attempt policy reliability. The study also explores AgileX Cobot Magic and Unitree G1; these extensions are distinct from the primary architecture comparison.

AMP1 studies a minimal manipulation interface with geometric tools and executable programs, reporting 104/108 successful real-robot trials with robot-error recovery and 94/108 without that recovery [211]. Its five-minute execution budget excludes model reasoning time, and collision checks are available tools rather than mandatory admission gates. The result establishes manipulation performance under those conditions; real-time safety would require enforced checks and full latency accounting. InstAgent studies retained program experience across G1, Go2, and ALOHA, with the initial and evolved experience libraries both frozen before the measured trials [40]. This evaluates reuse of revised experience around fixed models and skills, with the experience library held constant during testing.

Laboratory Automation Laboratory tasks connect model reasoning to devices whose interfaces and operating conditions are explicitly documented. Coscientist combines GPT-4 with documentation retrieval, code execution, and experimental automation [264]. It uses laboratory APIs and an Opentrons liquid handler to carry out experiments, including chemical reaction studies. The runtime contribution lies in connecting a proposed experiment to executable device instructions and then using experimental results in subsequent decisions. This provides a concrete application of Digital Resources and device-grounded execution without requiring a humanoid or mobile robot.

The automated reasoning loop does not cover every material transfer. Coscientist's integrated chemical setup required manual plate transfer, even when the system selected experimental conditions [264]. Autonomy in this setup is therefore interrupted at the plate-transfer stage. BioAgent extends this setting with teachable skill packages and bounded recovery for two long-horizon laboratory protocols [228]. In the source paper's Table II, with 50 runs per task and condition, enabling the reported recovery mechanism raises completion

from 27/50 to 33/50 on one protocol and from 20/50 to 29/50 on the other. Skill teaching, policy adaptation, and evaluation-time recovery remain distinct contributions; new recovery skills are not learned during those evaluation runs. Self-Building Labs combines experiment planning, digital-twin construction, device interfaces, and safety checks in physical chemistry demonstrations [229]. Human preparation and supervision remain part of those demonstrations, and its cross-workstation robot coordination is shown in simulation rather than demonstrated as an autonomous physical laboratory network.

Inspection and Field Operation Inspection systems connect navigation and scene interpretation to a response outside the robot. HROS implements a quadruped inspection application with localization, motion planning, multimodal analysis, and alarm reporting [35]. Its residential-property experiments evaluate the deployed navigation and inspection loop. The architecture also describes memory and controlled updates, but those descriptions do not by themselves establish long-term improvement.

RegenHarness builds on the HROS setting with explicit evidence gates and demonstrates warehouse and closed-route quadruped cases [165]. It examines when observed events can advance accepted task state and how unresolved evidence affects continuation. Its evaluation covers the recorded inspection sequence and a navigation circuit, with repeated operation across deployments still untested. Its proposed update process is discussed separately in Section 8.4.

Field operation introduces uncertainty about both the environment and the resources available for reasoning. SPINE has been deployed on air and ground robots for language-guided missions in large, unstructured environments [265]. The reported deployments require exploration and navigation under sensor and compute constraints. The robot must acquire planning context while operating, and the associated work on smaller onboard models addresses the compute limits of these missions.

Multi-Robot Operation and Physical Experimentation Multi-robot systems use shared operation for different purposes. AutoRT organizes foundation-model-guided data collection across mobile manipulators, combining autonomous behavior with human supervision and teleoperation [25]. Physical Agentic AI instead assigns typed skills to mission stages and enforces cross-robot constraints before dispatch [54]. Its comparisons use Gazebo execution and mocked hardware adapters; two G1–Go2 demonstrations show direct handover and coordinated delivery.

ENPIRE uses multiple robots to test revisions through reset, rollout, verification, and policy-code improvement [199]. Its environment-construction stage uses human feedback before the resulting interfaces support automated experimentation. In real-world Push-T, increasing the fleet from one to eight agent-controlled YAM stations reduces the reported time to a normalized score of 1.0 from roughly five to two hours under shared environment interfaces, reward functions, and reset mechanisms. This is a time-to-target comparison, not a single-attempt success rate; the study also finds lower per-robot utilization and disproportionately higher token costs as the fleet grows. Prepared reset and verification facilities therefore enable faster experimentation while remaining part of its cost and operating assumptions.

SimEX constructs a reusable toolbox through simulation and then revises it using a small physical adaptation budget [221]. Its reported hardware test contains three YAM tasks with ten evaluation trials each, after five adaptation trials per task. The reported adaptation budget is approximately ten minutes of physical interaction per task; it excludes the full simulation search, reasoning, and reset cost. Robo-COP instead updates both orchestration memory and policy parameters, testing frozen post-adaptation configurations [49]. The uneven gains across three physical tasks support joint adaptation within that protocol; isolating the effect of memory revision requires further controls.

Evaluation must distinguish what a metric measures from what a comparison can establish. CoCoBench exposes coordination costs that completed tasks can hide, while LIBERO-RECOVER tests continuation from failed states rather than success from nominal initialization [47, 85]. These protocols expose specific difficulties, but the scores alone do not establish the benefit of a runtime mechanism. Thea provides a more direct comparison of execution organization by holding the decision model and underlying tools fixed [89]. SERA and InstAgent add fixed-model comparisons with selected experience or guidance frozen before measured evaluation; SafeHarness adds a matched runtime comparison with an explicit safe-success metric [39, 40, 41]. Task-level success can therefore provide evidence about a Harness design. Thea's retry conditions still limit

interpretation as single-attempt reliability. Robo-Harness K1 illustrates the remaining attribution problem: its richer interface changes perception tools together with memory, motion support, and completion feedback [129]. A whole-system gain therefore cannot be assigned to one added tool without a corresponding controlled comparison.

Hardware applications provide evidence about integrated operation under the reported physical conditions. Table 2 identifies the platforms, preparation, and human support that define this scope. ENPIRE demonstrates repeated physical experimentation under prepared reset and verification interfaces [199]. It does not by itself establish that retained revisions improve arbitrary later tasks. The handoff analyses in Section 3.2 likewise show why successful individual capabilities can leave an invalid continuation. These findings connect the mechanism comparisons to unresolved questions about composition and sustained improvement. Interpreting either requires outcome evidence independent of the runtime verdict being assessed; a more permissive evaluator must not count as better physical execution. The next section also draws on the timing and distributed-execution dependencies identified in the preceding sections.

Table 2 Physical application studies and the conditions supporting their claims. Hardware tests, demonstrations, human preparation, and simulated extensions are distinguished; no row implies implementation of all six responsibilities.

System and task	Physical platform	Runtime mechanism	Validation scope and human role
OK-Robot: household pick-and-drop [128]	Hello Robot Stretch mobile manipulator	Object localization, navigation, and grasp integration	Real-home trials; human-guided initial scan; static-map dependence
Thea: object collection and delivery [89]	Astribot S1; Cobot Magic and G1 extensions	Scene context, tool composition, and outcome evaluation	Quantitative architecture comparison on S1; additional platform demonstrations; human outcome annotation and task-level user interaction
Coscientist: chemical experiments [264]	Opentrons liquid handler and analytical instruments	Documentation-guided code and experimental feedback	Physical experiments; manual plate transfer in the integrated chemical setup
HROS / Argos: property inspection [35]	Vbot quadruped with LiDAR and cameras	Navigation, image analysis, and alarm reporting	Residential-property trials; alerts and corrections handled by human reviewers; no longitudinal update evaluation
SPINE: language-guided field missions [265]	UAVs and ground robots	Planning with exploration and constrained compute	Reported outdoor missions; operator intervention frequency not reported
AutoRT: fleet data collection [25]	Mobile manipulators	Task generation, allocation, and supervision	Physical fleet collection with human oversight and teleoperation
Physical Agentic AI: handover and delivery [54]	Unitree G1 and Go2	Typed skills and cross-robot dispatch checks	Two hardware demonstrations; quantitative comparisons use simulation or mock adapters; human requester, intervention rate not reported

Table 2. Cont.

System and task	Physical platform	Runtime mechanism	Validation scope and human role
ENPIRE: policy-improvement experiments [199]	Bimanual YAM stations, up to eight	Automated reset, verification, rollout, and code revision	Repeated physical trials; human feedback constructs the task environment before automated improvement
AMP1: general manipulation [211]	Franka arms; three rigs	Minimal SDK, geometric tools, programs, and robot-error recovery	Twelve tasks, 108 trials per condition; 104/108 with controller-error recovery, 94/108 without; five-minute budget excludes reasoning
InstAgent: heterogeneous collaboration [40]	Unitree G1, Go2, and ALOHA	Behavior trees and retained program experience	20 tasks and three models; initial / evolved libraries frozen before testing; prepared skills and human task interaction
BioAgent: laboratory protocols [228]	Monte 02 dual-arm robot	Teachable skill packages and bounded recovery	Two protocols, 50 runs per protocol and condition; human teaching and protocol review; no new recovery-skill learning during evaluation
Self-Building Labs: chemistry automation [229]	Laboratory robots and scientific instruments	Digital twins, device APIs, workflow and safety checks	Physical chemistry demonstrations with human setup and supervision; cross-workstation robot coordination is simulated
SpatialHarness: fine manipulation [104]	Franka FR3	Synchronized spatial scene, virtual views, and pose information	Four tasks, 60 trials per condition; 11/60 to 50/60 successes for the full spatial package; setup cost not captured by policy-query counts
BT-Agent: onboard task execution [162]	Boston Dynamics Spot	Behavior trees, shared state, and independent guards	Five tasks, three trials each; 11/15 completions; human-approved motion and manually corrected map; command latency is not physical stopping time
SimEX: toolbox adaptation [221]	YAM robot	Simulation-built toolbox and physical-feedback repair	Three tasks, ten final trials each; 26/30 successes after five adaptation trials per task; human resets and prior simulation search

10 Open Problems and Research Directions

The reviewed systems expose dependencies across Harness responsibilities that remain difficult to manage. A skill may finish without preparing its successor, and a useful revision may invalidate assumptions elsewhere in the runtime. The following directions examine these dependencies and the evidence needed to assess their effects on physical execution.

Model–Harness Co-design Model–Harness co-design concerns which behavior is learned by the model and which is organized explicitly by the runtime. Guava explores workflow and interface design alongside training a compact model [27]. Harness VLA instead uses external guidance to extend a frozen action policy [26]. These choices affect where task knowledge is stored and where execution can be corrected. The division also depends on what information the model can use. In 2AM and AR-WAM, structured agent guidance is paired with action-model training [140, 141]. An interface that works with an adapted model may therefore be

unsuitable for an arbitrary frozen policy. AMP1 and EmbodiedSkillBench show that the benefit of interface assistance depends on the model and on whether the interface restricts the actions available to it [211, 257]. TinyAstral separates harness selection around a fixed teacher from subsequent student training, whereas A2-MEAS and HITCH train action or tracking interfaces that the surrounding agent uses [151, 150, 149]. Robo-COP changes policy weights alongside orchestration memory [49]. Factorial comparisons of model, interface, and adaptation budget could help attribute these gains.

Comparisons should separate interface changes from model adaptation wherever the interfaces remain compatible. Holding the model and available capabilities fixed can help isolate a runtime change outside the capability library. When the library itself is changed, the comparison should report that change and its preparation cost. When adaptation is required, matched training data and compute budgets can help identify its additional contribution. Such studies should examine where errors are detected and corrected, alongside final task performance. Tests under changed observations or task conditions could then establish whether the chosen division remains effective beyond its original setting.

Reliable Capability Composition Reliable capability composition depends on the states passed between successive skills. Semantic handoff analyses show that a skill can meet its own completion condition while leaving the next skill unable to start reliably [74]. BATON addresses these transitions through transition-aware memory [75]. The remaining difficulty is to determine which physical conditions matter for a particular transition and whether the available evidence establishes them. For example, a pose estimate may appear acceptable while contact or object support remains unsuitable. Compatible invocation interfaces alone cannot establish readiness for the next skill.

Continuous skill chains provide a basis for learning these conditions. Successor skills could be trained on states actually produced by their predecessors. Transition checks could identify when another observation or a preparatory action is needed. Such checks should distinguish an unsuitable physical state from insufficient evidence about that state. Evaluation should preserve the states left by preceding skills and account for the cost of any corrective action. Resetting each skill to a valid initial state would remove the dependency that the method is intended to address.

Timely Decisions under Physical Uncertainty Runtime decisions need evidence that remains usable when the resulting action takes effect. Observation timing therefore needs to be considered alongside reasoning latency and the duration of committed actions. AESOP accommodates slow reasoning within a fast control process [15]. Kairos coordinates inference scheduling with executed action horizons [121]. These mechanisms address different parts of the delay between observing a state and acting on it. More reasoning can improve a proposal while allowing its supporting evidence to become stale. A longer action interval can reduce waiting for inference but delay correction at the model level, even when lower-level feedback continues.

The research problem is to choose these time scales jointly as physical uncertainty changes. An additional observation or a longer reasoning interval is useful only if its expected benefit justifies the delay. This trade-off should be studied within the interruption limits of the invoked capability. Evaluation needs to relate decision quality to evidence age and the time until an intervention takes physical effect. When an intervention deadline is shorter than model latency, local protection must remain available while reasoning continues. The resulting system also needs a way to reconcile the model's eventual response with any intervention already taken.

Cross-Robot and Distributed Execution Distributed execution requires the runtime to determine which execution assumptions remain valid when computation or task responsibility moves. Moving inference to another service can leave the robot and task owner unchanged. Kairos illustrates how shared serving depends on the robot's execution process [121]. If a service changes or communication is interrupted, a returned result may no longer match the robot's current state. Research is needed on how to associate results with the observations and requests that produced them, and when to reject or recompute them. Tests should include delayed responses and interrupted communication, with physical continuation assessed alongside service recovery.

Transferring a subtask to another robot introduces constraints on embodiment and execution authority. Physical Agentic AI makes robot-specific skills and dispatch conditions explicit [54]. A transfer protocol must additionally account for operations still active under the previous assignment. Updating the task owner does

not resolve actions still running on the original robot. An open question is how to transfer valid progress while obtaining new evidence for conditions that the receiving robot cannot inherit. Evaluation should examine whether transfers avoid conflicting actions under communication loss and differences in device semantics.

Sustained Improvement from Physical Experience Evaluating sustained improvement requires tracking later benefits alongside regressions in previously supported behavior. SHAPER studies skill and Harness revision in simulation [36]. ENPIRE supports repeated hardware experiments through prepared reset and verification facilities [199]. Recent studies add more explicit separation between development and evaluation: SERA tests new instances of the same tasks, Recursive Harness Distillation freezes the refined playbook before evaluation, and SEAR tests inherited support after budgeted development [39, 218, 48]. These results establish bounded forms of reuse; they do not yet show sustained gains across arbitrary task sequences, changing hardware, and repeated releases. A revision can repair its motivating failure while disrupting a skill transition that previously worked. Limited physical trial budgets make such regressions difficult to detect before adoption.

One direction is to select regression trials using the dependencies affected by a revision. This can focus testing on likely failures, but it should be supplemented with independent tests outside the update-selection loop. Otherwise, the procedure can overlook interactions absent from its dependency records. Longitudinal evaluation should track whether gains survive later episodes and further updates. Reuse on the same task and transfer to new tasks should be reported separately. When the evaluator also changes, outcome assessment needs an independent reference to distinguish better execution from more permissive judgments. The cost of maintaining this evidence is part of the improvement process and should be reported with its gains.

11 Conclusion

This survey defines the Physical Agentic Harness through six runtime responsibilities and uses them to compare over 200 works across robot architectures, skill frameworks, runtime assurance, and foundation-model systems. The comparison traces how model decisions reach physical execution and how observed outcomes inform later decisions in both single-robot and multi-robot systems.

The evidence now includes controlled improvements from selected runtime changes around fixed models, alongside broader comparisons that change skills, information access, or training. The central evaluation problem is to identify which mechanism helps under which physical conditions. Comparisons need to identify the intended change and control the other relevant conditions. Changes to capability libraries, observation access, retry budgets, or development costs must be reported, with task success distinguished from progress, cumulative recovery, and safe completion. For persistent evolution, frozen post-adaptation tests support bounded reuse, while sustained improvement requires evidence that retained revisions remain useful through later tasks and updates. Revising the updater itself raises a further question: whether that revision improves subsequent update cycles under independent evaluation.

Author List

Chenxi Zeng[1,3†], Zihan Guo[1,4†], Zicai Cui[1,2†], Yang Tang[1,2†], Yue Xin[1,5†], Zhi Han[1,2†], Liuhaichen Yang[1†], Shao Zhang[1], Chen Zhang[1], Yishuai Cai[1], Chao Wu[1], Bingqi Jiang[1], Jingjing Zhang[1], Dongzhan Zhou[1], Shuyue Hu[1], Ming Zhou[1], Weinan Zhang[1,2], Yang Li[1,2]

[1]Shanghai AI Laboratory, [2]Shanghai Jiao Tong University, [3]Jilin University, [4]Sun Yat-sen University, [5]Beihang University

Correspondence: Weinan Zhang (wnzhang@sjtu.edu.cn) and Yang Li (yang.li.cs@sjtu.edu.cn)
First author contact: Chenxi Zeng (zengcx2423@mails.jlu.edu.cn)

[†]These authors contributed equally to this work and share first authorship.

References

[1] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2165–2183. PMLR, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

[2] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An Open-Source Vision-Language-Action Model. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 2679–2713. PMLR, 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

[3] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, 2025. doi: 10.15607/RSS.2025.XXI.010. URL https://doi.org/10.15607/RSS.2025.XXI.010.

[4] Gemini Robotics Team. Gemini robotics: Bringing AI into the physical world. *arXiv preprint arXiv:2503.20020*, 2025. URL https://arxiv.org/abs/2503.20020.

[5] 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.

[6] Yuheng Ji, Huajie Tan, Jiayu Shi, Xiaoshuai Hao, Yuan Zhang, Hengyuan Zhang, Pengwei Wang, Mengdi Zhao, Yao Mu, Pengju An, Xinda Xue, Qinghang Su, Huaihai Lyu, Xiaolong Zheng, Jiaming Liu, Zhongyuan Wang, and Shanghang Zhang. RoboBrain: A unified brain model for robotic manipulation from abstract to concrete. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1724–1734, 2025.

[7] NVIDIA, Alisson Azzolini, et al. Cosmos-Reason1: From physical common sense to embodied reasoning. *arXiv preprint arXiv:2503.15558*, 2025.

[8] NVIDIA, Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, Prithvijit Chattopadhyay, Yongxin Chen, Yin Cui, Yifan Ding, et al. Cosmos world foundation model platform for physical AI. *arXiv preprint arXiv:2501.03575*, 2025.

[9] Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, et al. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

[10] Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi Fan, and Joel Jang. World Action Models are Zero-shot Policies. *arXiv preprint arXiv:2602.15922*, 2026. doi: 10.48550/arXiv.2602.15922. URL https://arxiv.org/abs/2602.15922.

[11] MotuBrain Team. MotuBrain: An advanced world action model for robot control. *arXiv preprint arXiv:2604.27792*, 2026.

[12] GigaWorld Team, Angen Ye, Angyuan Ma, Boyuan Wang, Chaojun Ni, et al. GigaWorld-Policy-0.5: A faster and stronger WAM empowered by AutoResearch. *arXiv preprint arXiv:2607.13960*, 2026.

[13] Lucy Xiaoyang Shi, Brian Ichter, Michael Robert Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 54919–54933. PMLR, 2025.

[14] 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 *Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 1769–1782. PMLR, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

[15] Rohan Sinha, Amine Elhafsi, Christopher Agia, Matt Foutter, Edward Schmerling, and Marco Pavone. Real-Time Anomaly Detection and Reactive Planning with Large Language Models. In *Robotics: Science and Systems XX*, 2024. doi: 10.15607/rss.2024.xx.114. URL https://doi.org/10.15607/rss.2024.xx.114.

[16] Zeyi Liu, Arpit Bahety, and Shuran Song. Reflect: Summarizing robot experiences for failure explanation and correction, 2023. URL https://arxiv.org/abs/2306.15724.

[17] Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. AHA: A Vision-Language-Model for Detecting and Reasoning Over Failures in Robotic Manipulation. *International Conference on Learning Representations*, 2025:45493, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/70a06501001e1820fd1eb9ee821302d2-Abstract-Conference.html.

[18] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

[19] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

[20] Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026. URL https://arxiv.org/abs/2603.28052.

[21] Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. AutoHarness: Improving LLM agents by automatically synthesizing a code harness, 2026. URL https://arxiv.org/abs/2603.03329.

[22] Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-Harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498v3*, 2026. doi: 10.48550/arXiv.2606.09498. URL https://arxiv.org/abs/2606.09498v3.

[23] Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-Time Execution of Action Chunking Flow Policies. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1122. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/300ccb2187dedd4edcc07f7e76d8e553-Abstract-Conference.html.

[24] Roya Firoozi, Johnathan Tucker, Stephen Tian, Anirudha Majumdar, Jiankai Sun, Weiyu Liu, Yuke Zhu, Shuran Song, Ashish Kapoor, Karol Hausman, Brian Ichter, Danny Driess, Jiajun Wu, Cewu Lu, and Mac Schwager. Foundation Models in Robotics: Applications, Challenges, and the Future. *The International Journal of Robotics Research*, 44(5):701–739, 2025. doi: 10.1177/02783649241281508. URL https://doi.org/10.1177/02783649241281508.

[25] Michael Ahn, Debidatta Dwibedi, Chelsea Finn, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Karol Hausman, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Sean Kirmani, Isabel Leal, Edward Lee, Sergey Levine, Yao Lu, Isabel Leal, Sharath Maddineni, Kanishka Rao, Dorsa Sadigh, Pannag Sanketi, Pierre Sermanet, Quan Vuong, Stefan Welker, Fei Xia, Ted Xiao, Peng Xu, Steve Xu, and Zhuo Xu. AutoRT: Embodied Foundation Models for Large Scale Orchestration of Robotic Agents. *arXiv preprint arXiv:2401.12963*, 2024. URL https://arxiv.org/abs/2401.12963.

[26] Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Yi Nie, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Jiaji Rao, Zhengru Fang, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness VLA: Steering Frozen VLAs into Reliable Manipulation Primitives via Memory-Guided Agents. *arXiv preprint arXiv:2607.08448v5*, 2026. URL https://arxiv.org/abs/2607.08448v5.

[27] Haowen Liu, Xirui Li, Shaoxiong Yao, Peng Shi, Tianyi Zhou, Jia-Bin Huang, Furong Huang, and Jiayuan Mao. Guava: Distilling Frontier VLMs into a Compact Agent through a Robotic Manipulation Harness. *arXiv preprint arXiv:2606.18363*, 2026. URL https://arxiv.org/abs/2606.18363v3.

[28] Erann Gat. On three-layer architectures. In David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, pages 195–210. AAAI Press, 1998.

[29] Matthias Mayr, Francesco Rovida, and Volker Krueger. SkiROS2: A skill-based Robot Control Platform for ROS. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6273–6280, 2023. doi: 10.1109/IROS55552.2023.10342216. URL https://doi.org/10.1109/IROS55552.2023.10342216.

[30] Ankush Desai, Shromona Ghosh, Sanjit A. Seshia, Natarajan Shankar, and Ashish Tiwari. SOTER: A Runtime Assurance Framework for Programming Safe Robotics Systems. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 138–150, 2019. doi: 10.1109/dsn.2019.00027. URL https://doi.org/10.1109/dsn.2019.00027.

[31] Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR, 2023. URL https://proceedings.mlr.press/v205/ichter23a.html.

[32] 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 *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500, 2023. doi: 10.1109/ICRA48891.2023.10160591. URL https://doi.org/10.1109/ICRA48891.2023.10160591.

[33] Yanzhe Chen, Zechen Bai, Zhijun Cao, Wenzheng Zeng, Kevin Qinghong Lin, Yiqi Lin, Guoqiang Liang, Kevin Yuchen Ma, Qiming Huang, and Mike Zheng Shou. Show-Harness: Just a VLM agent can play robots. *arXiv preprint arXiv:2609.10522*, 2026. URL https://arxiv.org/abs/2609.10522.

[34] Guojun Chen, Alex Schott, and Lin Zhong. TypeGo: An OS runtime for embodied agents. *arXiv preprint arXiv:2607.05482*, 2026. URL https://arxiv.org/abs/2607.05482.

[35] Yaoyuan Yan, Zhiyou Heng, Haoxiang Jie, Gang Liu, Hongjie Yan, and Wei Zhou. Harness Robotic OS: A Unified Embodied-Agent Runtime for Closed-Loop Quadruped Inspection. *arXiv preprint arXiv:2609.11225*, 2026. URL https://arxiv.org/abs/2609.11225.

[36] Peidong Wang, Zhiming Ma, Ying Chang, Xufang Luo, Yiqun Zhang, Zihan Wang, Xiaocui Yang, Shi Feng, Yuqing Yang, and Dongsheng Li. Self-Evolving Embodied Agents via Skill-Harness Evolution, 2026. URL https://arxiv.org/abs/2608.11350v2.

[37] Xin Ding, Liang Mi, Mingzhe Huang, Zixuan Wang, Chao Zhang, Zixu Hao, Fu Chen, Xiangyu Li, Yikai Zheng, Yaoyu Guo, Weijun Wang, Kun Li, Hao Wu, Yunxin Liu, and Ting Cao. Zetta ζ: An Efficient Closed-Loop Embodied Harness for Self-Evolving Physical Intelligence. *arXiv preprint arXiv:2608.16590*, 2026. URL https://arxiv.org/abs/2608.16590.

[38] Sanghoon Lee, Jiyeong Chae, and Kyung-Joon Park. Harness Engineering for Physical AI: Robot Middleware Is the Harness Layer. *arXiv preprint arXiv:2606.09416*, 2026. URL https://arxiv.org/abs/2606.09416.

[39] Tingwen Du, Haoyu Guo, Hanxue Zhang, Xuanxu Wang, Junting Dong, Chao Gao, Mingfei Han, Tao Lu, Mulin Yu, Chunhua Shen, Xiaojun Chang, and Zhihui Li. From VLM Planners to Robotic Agents: Self-Evolving Harnesses for History-Dependent Manipulation, 2026. URL https://arxiv.org/abs/2610.29560. arXiv preprint arXiv:2610.29560.

[40] Haoyang Luo, Junhao Qin, Jingyan Cao, Yi Huang, Guangjie Yang, Xiaoyu Chen, Haotao Lu, Tianxiang Gui, Ye Shi, and Jingya Wang. InstAgent: A Recursively Self-Improving Agent Operating System for Heterogeneous Robot Collaboration, 2026. URL https://arxiv.org/abs/2610.23098. arXiv preprint arXiv:2610.23098.

[41] Bingxin Xu, Yuzhang Shang, Zhen Dong, and Emilio Ferrara. Coding Agents with Harness for Safe Robot Control, 2026. URL https://arxiv.org/abs/2609.20822. arXiv preprint arXiv:2609.20822.

[42] Shaunak A. Mehta, Ananya Hazarika, Haochen Zhang, Fan Yang, Ryo Moriyama, Wenkai Li, Yash Patel, and Kanata Suzuki. Toward Unified Robot Learning: Bridging Representation, Vision-Language-Action, and World Models. *arXiv preprint arXiv:2609.03927*, 2026. URL https://arxiv.org/abs/2609.03927.

[43] Yukiyasu Domae, Keisuke Shirai, Hanbit Oh, Ryoichi Nakajo, Tomohiro Motoda, Koshi Makihara, Masaki Murooka, Takuma Yagi, Yoshiaki Bando, and Ryo Hanai. The Embodiment Gap in Robot Foundation Models. *arXiv preprint arXiv:2608.18433*, 2026. URL https://arxiv.org/abs/2608.18433.

[44] Zuxing Lu, Hongjia Zhai, Guanzhi Wang, Huajian Zeng, Jiaqi Yang, Jingyu Liu, Lei Cheng, Yuantai Zhang, Yuheng Qiu, Zezhou Cheng, Ivan Laptev, Danfei Xu, Benjamin Riviere, Giuseppe Loianno, Eric Xing, and Xingxing Zuo. World-Action Models for Robot Learning and Control: A Survey. *arXiv preprint arXiv:2609.16074*, 2026. URL https://arxiv.org/abs/2609.16074.

[45] Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, Congming Zheng, Jiachen Zhu, Zeyu Zheng, Zhuosheng Zhang, Xingyu Lou, Changwang Zhang, Zhihui Fu, Jun Wang, Weiwen Liu, Jianghao Lin, and Weinan Zhang. Externalization in LLM Agents: A Unified Review of Memory, Skills, Protocols and Harness Engineering. *arXiv preprint arXiv:2604.08224*, 2026. URL https://arxiv.org/abs/2604.08224.

[46] Qianyu Meng, Yanan Wang, Liyi Chen, Yihang Li, Wei Wu, Wenyuan Jiang, Qimeng Wang, Chengqiang Lu, Yan Gao, Yi Wu, and Yao Hu. Agent Harness for Large Language Model Agents: A Survey. *Preprints*, 2026. doi: 10.20944/preprints202604.0428.v3. URL https://www.preprints.org/manuscript/202604.0428/v3.

[47] Yang Chen, Ye-Xin Xie, Lirong Che, Danyang Peng, Yuzhe Yang, Peiwen Lin, Xu Cao, Chuang Wang, Lei Yuan, Jian Su, and Lan-Zhe Guo. Cocobench: A cooperative coordination benchmark for embodied multi-agent task planning, 2026. URL https://arxiv.org/abs/2608.28266.

[48] Xinyi Wang, Bangzheng Li, Hanyu Wang, Lin Zhang, Andrea Yaoyun Cui, Zhenggang Tang, Shyamal Buch, and Dejia Xu. SEAR: Self-Evolving Agentic Robotics, 2026. URL https://arxiv.org/abs/2610.18792. arXiv preprint arXiv:2610.18792.

[49] Xilun Zhang, Maggie Wang, Erik Bauer, Hong-Xing Yu, Huang Huang, Jiajun Wu, and Marco Pavone. Co-Evolving Robot Orchestrators and Policies through Deployment, 2026. URL https://arxiv.org/abs/2610.09228. arXiv preprint arXiv:2610.09228.

[50] Maj Stenmark. Industrial Robot Skills. In Manfred Jaeger, Thomas Dyhre Nielsen, and Paolo Viappiani, editors, *Frontiers in Artificial Intelligence and Applications*, volume 257, pages 295–298. IOS Press, 2013. doi: 10.3233/978-1-61499-330-8-295. URL https://doi.org/10.3233/978-1-61499-330-8-295.

[51] Tully Foote. tf: The transform library. In *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–6, 2013. doi: 10.1109/tepra.2013.6556373. URL https://doi.org/10.1109/tepra.2013.6556373.

[52] Paul Furgale, Joern Rehder, and Roland Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1280–1286, 2013. doi: 10.1109/iros.2013.6696514. URL https://doi.org/10.1109/iros.2013.6696514.

[53] Yanjiang Guo, Yen-Jen Wang, Lihan Zha, and Jianyu Chen. DoReMi: Grounding Language Model by Detecting and Recovering from Plan-Execution Misalignment. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12124–12131, 2024. doi: 10.1109/IROS58592.2024.10802284. URL https://doi.org/10.1109/IROS58592.2024.10802284.

[54] Xinyuan Liu, Eren Sadikoglu, Riana Chatterjee, and Ransalu Senanayake. Physical Agentic AI: An Architecture for Orchestrating a Robot Crew with LLMs. *arXiv preprint arXiv:2608.22657*, 2026. URL https://arxiv.org/abs/2608.22657.

[55] Shuhan Guo, Kun Zhang, Haifei Liu, Xingyu Gao, Yongqi Zhang, Yaqing Wang, and Quanming Yao. ContextFlow: Hierarchical Task-State Alignment for Long-Horizon Embodied Agents. *arXiv preprint arXiv:2605.19314*, 2026. URL https://arxiv.org/abs/2605.19314.

[56] Yang Liu, Weixing Chen, Xinshuai Song, Tao Pu, Siwen Mo, Yongjie Bai, Zihao Chen, Qianran Sun, Liruo Zhong, Ying Shen, and Liang Lin. PhyAgentOS: A Self-Evolving Operating System for Embodied Agents with Decoupled Cognitive Planning and Physical Execution. *arXiv preprint arXiv:2607.16636*, 2026. URL https://arxiv.org/abs/2607.16636.

[57] Disha Kamale and Dmitry Berenson. Pddl-art: Autonomous symbolic abstraction from demonstration for long-horizon robotic manipulation using vision-language models. *arXiv preprint arXiv:2608.17146*, 2026. URL https://arxiv.org/abs/2608.17146.

[58] Mohammad Albinhassan, Yuming Feng, Alessandra Russo, and Pranava Madhyastha. Neurosymbolic embodied agents, 2026. URL https://arxiv.org/abs/2608.16794.

[59] Xiao Wang, Lu Dong, Ifeoma Nwogu, Srirangaraj Setlur, and Venu Govindaraju. MistyPilot: Enabling Social-Robot Control through Multi-Agent LLM Skill Orchestration. *arXiv preprint arXiv:2608.15549*, 2026. URL https://arxiv.org/abs/2608.15549.

[60] Sayem Nazmuz Zaman and Cyrus Neary. When automata meet streams: Temporal logic compilation for stream-based robotics task and motion planning. *arXiv preprint arXiv:2608.19453*, 2026. URL https://arxiv.org/abs/2608.19453.

[61] Weiqi Wang, Zhi Li, Yudong Lei, David Martinez, Xiaofeng Gao, Yuxin Jiang, Chenfanfu Jiang, Yingnian Wu, Demetri Terzopoulos, and Ran Gong. Sun: Persistent programs for language-grounded control-to-learning-to-real policies, 2026. URL https://arxiv.org/abs/2608.31167v1.

[62] Utkarsh A. Mishra, Alperen Degirmenci, Yongxin Chen, Danfei Xu, Tucker Hermans, and Caelan Garrett. TAMPAgent: Agentic Belief-Space Planning for Mobile Manipulation, 2026. URL https://arxiv.org/abs/2611.03584. arXiv preprint arXiv:2611.03584.

[63] Enqiao Lu, Yuxiang Xie, Xingrui Yu, Yuyang Wang, Yipeng Gu, Zhirui Fang, Xinrun Xu, Zhenglin Wan, Pengfei Zhou, Wangbo Zhao, Yang You, Yuxiang Gao, and Ivor Tsang. LOGIV: Graph-Verified Execution with Causal Local Repair for Long-Horizon Robot Tasks, 2026. URL https://arxiv.org/abs/2610.40016. arXiv preprint arXiv:2610.40016.

[64] Meiqi Wang and Shichao Li. Compiling and Benchmarking Task-State Horizons for Embodied Agents. *arXiv preprint arXiv:2608.08036*, 2026. URL https://arxiv.org/abs/2608.08036.

[65] Wei Wang, Wenqiao Zhang, Yutong Lin, Yuqian Yuan, Tianwei Lin, Jinhao Mao, Zhenxuan Fan, Mingjian Gao, Yang Dai, Wentong Li, Zheqi Lv, Zheng Dong, Yingjie Niu, Jiaqi Zhu, Jun Xiao, Chao Li, and Yueting Zhuang. EmbodiedSkills: A Unified Framework for Orchestrating, Training, and Deploying VLA Agents. *arXiv preprint arXiv:2609.01281v1*, 2026. doi: 10.48550/arXiv.2609.01281. URL https://arxiv.org/abs/2609.01281v1.

[66] Tsunehiko Tanaka, Matthew Stephenson, Alistair Macvicar, and Edgar Simo-Serra. Evidence-gated task and motion planning with vision-language models, 2026. URL https://arxiv.org/abs/2608.20084.

[67] Puru Ojha, Narendhiran Vijayakumar, Nav Singhal, Girish Varma, and Antony Thomas. Plan along the way: Event-triggered foundation-model planning for tamp execution in partially observable manipulation, 2026. URL https://arxiv.org/abs/2608.28075v1.

[68] Brian P Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International journal of robotics research*, 23(9):939–954, 2004. doi: 10.1177/0278364904045564. URL https://doi.org/10.1177/0278364904045564.

[69] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International journal of robotics research*, 32(12):1495–1512, 2013. doi: 10.1177/0278364913496484. URL https://doi.org/10.1177/0278364913496484.

[70] Han-Lim Choi, Luc Brunet, and Jonathan P How. Consensus-based decentralized auctions for robust task allocation. *IEEE transactions on robotics*, 25(4):912–926, 2009. doi: 10.1109/TRO.2009.2022423. URL https://doi.org/10.1109/TRO.2009.2022423.

[71] Daniel Garces, Sara Castro, Adrian Haimovich, Byron Crowe, and Stephanie Gil. Model-based reinforcement learning for heterogeneous multi-robot task assignment under distribution shifts. *arXiv preprint arXiv:2608.21554*, 2026. URL https://arxiv.org/abs/2608.21554.

[72] Pengfei Zhou, Yixing Ma, Zhengfeiyang Zhang, Haoyuan Chen, Zhenglin Wan, Hexin Wang, Wangbo Zhao, and Yang You. REA: A Routed Embodied Agent Runtime for Asynchronous Brain-Cerebellum Control, 2026. URL https://arxiv.org/abs/2610.42053. arXiv preprint arXiv:2610.42053.

[73] Haoyuan Deng, Jiebin Liu, Tengxiao Zhang, Langning Yan, Hongye Cao, and Ziwei Wang. DynaHarness: A Dynamic Physical Harness for Self-Evolving Robot Agents, 2026. URL https://arxiv.org/abs/2609.40306. arXiv preprint arXiv:2609.40306.

[74] Ke Rui, Yushen Zuo, Jiawei Wang, Haoran Jia, Jinming Ma, Weitao Zhou, and Minglei Li. Diagnosing Semantic Handoff Failures in Agent-Orchestrated Vision-Language-Action Skill Composition. *RSS SemRob Workshop 2026*, 2026. URL https://arxiv.org/abs/2607.06256.

[75] Bingxin Xu, Yuzhang Shang, and Emilio Ferrara. Don't drop the baton: Long-horizon robot manipulation via agentic subtask exploration and transition-aware memory, 2026. URL https://arxiv.org/abs/2608.16889.

[76] Philip Huang, Ruixuan Liu, Shobhit Aggarwal, Changliu Liu, and Jiaoyang Li. Apex-mr: Multi-robot asynchronous planning and execution for cooperative assembly, 2025. URL https://arxiv.org/abs/2503.15836.

[77] Fan Yang, Soumya Teotia, Shaunak A. Mehta, Prajit KrisshnaKumar, Quanting Xie, Jun Liu, Yueqi Song, Wenkai Li, Atsunori Moteki, Kanji Uchino, and Yonatan Bisk. Indoorr2x: Indoor robot-to-everything coordination with llm-driven planning. *arXiv preprint arXiv:2603.20182*, 2026. URL https://arxiv.org/abs/2603.20182.

[78] Arjo Chakravarty, Michael X. Grey, M. A. Viraj J. Muthugala, and Mohan Rajesh Elara. Time-ordered ad-hoc resource sharing for independent robotic agents, 2024. URL https://arxiv.org/abs/2408.07942.

[79] Béatrice Garcia Cegarra, Elena Vanneaux, Quentin Picard, and David Filliat. Connectivity-aware graph extension for decentralized multi-robot exploration, 2026. URL https://arxiv.org/abs/2609.00804.

[80] Zeno Research. Zeno-1: Collaborative intelligence for robots that work together. Zeno AI, September 2026. URL https://www.zenobot.ai/research/zeno-1-collaborative-intelligence. Accessed: October 7, 2026.

[81] Enshen Zhou, Qi Su, Cheng Chi, Zhizheng Zhang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, and He Wang. Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6919–6929, 2025. doi: 10.1109/CVPR52734.2025.00649.

[82] Sihyung Yoon, Minjong Yoo, Sanghyun Ahn, Seojeong Choi, and Honguk Woo. RoboBRIDGE: A Modular Framework for Bridging Policies to Robust Real-World Robotic Agents. *arXiv preprint arXiv:2607.27881*, 2026. URL https://arxiv.org/abs/2607.27881.

[83] Ganlong Zhao, Zijia Tang, Xingping Chen, Zhanghui Kuang, Ye Tian, and Guanbin Li. Flare: A failure-aware framework for autonomous correction and recovery in visual-language robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22391–22401, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Zhao_FLARE_A_Failure-Aware_Framework_for_Autonomous_Correction_and_Recovery_in_CVPR_2026_paper.html.

[84] Sheng Xu, Ruixing Jin, Huayi Zhou, Bo Yue, Guanren Qiao, Yunxin Tai, Yueci Deng, Kui Jia, and Guiliang Liu. From reaction to anticipation: Proactive failure recovery through agentic task graph for robotic manipulation, 2026. URL https://arxiv.org/abs/2605.11951.

[85] Lin Liu, Zhicheng Bao, Lu Zhang, Ziying Song, Wu Yang, Yuzheng Zhuang, Shuai Tao, Wulong Liu, Caiyan Jia, and Huchuan Lu. LIBERO-RECOVER: Beyond Task Success Towards Failure Recovery in Robotic Manipulation Models, 2026. URL https://arxiv.org/abs/2609.05178v2.

[86] Moritz Tenorth and Michael Beetz. KnowRob: A knowledge processing infrastructure for cognition-enabled robots. *The International Journal of Robotics Research*, 32(5):566–590, 2013. doi: 10.1177/0278364913481635. URL https://doi.org/10.1177/0278364913481635.

[87] Antoni Rosinol, Arjun Gupta, Marcus Abate, Jingnan Shi, and Luca Carlone. 3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans. In *Robotics: Science and Systems XVI*, 2020. doi: 10.15607/rss.2020.xvi.079. URL https://doi.org/10.15607/rss.2020.xvi.079.

[88] Nathan Hughes, Yun Chang, and Luca Carlone. Hydra: A Real-time Spatial Perception System for 3D Scene Graph Construction and Optimization. In *Robotics: Science and Systems XVIII*, 2022. doi: 10.15607/rss.2022.xviii.050. URL https://doi.org/10.15607/rss.2022.xviii.050.

[89] Qi Wang, Tianyi Wang, Chengyang Li, Shikun Ban, Yurun Chen, Yizhong Ge, Jason Qin, Chengtai Li, and Wentao Zhu. Towards the Harness of Embodied Agents. *arXiv preprint arXiv:2608.11246*, 2026. URL https://arxiv.org/abs/2608.11246.

[90] Lei Fu, Sahar Salimpour, Leonardo Militano, Harry Edelman, Jorge Peña Queralta, and Giovanni Toffetti. ROSBag MCP Server: Analyzing Robot Data with LLMs for Agentic Embodied AI Applications. *arXiv preprint arXiv:2511.03497*, 2025. URL https://arxiv.org/abs/2511.03497.

[91] Jiaping Xiao, Pingyuan Ji, and Mir Feroskhan. ADM-Planner: LLM-Guided Long-Horizon Planning for Mobile Manipulators with Attention-Enhanced Dynamic Memory. *arXiv preprint arXiv:2609.29212*, 2026. URL https://arxiv.org/abs/2609.29212.

[92] Mingcong Lei, Honghao Cai, Yuyuan Yang, Yimou Wu, Jinke Ren, Zezhou Cui, Liangchen Tan, Junkun Hong, Gehan Hu, Shuangyu Zhu, Shaohan Jiang, Ge Wang, Junyuan Tan, Zhenglin Wan, Zheng Li, Zhen Li, Shuguang Cui, Yiming Zhao, and Yatong Han. RoboMemory: A Brain-inspired Multi-memory Agentic Framework for Interactive Environmental Learning in Physical Embodied Systems. *arXiv preprint arXiv:2508.01415*, 2025. URL https://arxiv.org/abs/2508.01415.

[93] Zijian Zeng, Fei Ding, Huiming Yang, and Xianwei Li. HELM: Harness-Enhanced Long-horizon Memory for Vision-Language-Action Manipulation. *arXiv preprint arXiv:2604.18791*, 2026. URL https://arxiv.org/abs/2604.18791.

[94] Junxiang Wang, Yuchen Liu, Yi Wu, Guofei Chen, Nir Pechuk, Tao Chen, Zackory Erickson, Yuzhe Qin, and Tianyu Li. What Should Embodied Agents Remember? Benchmarking and Evaluating Memory Modality for Embodied Agents, 2026. URL https://arxiv.org/abs/2610.12561. arXiv preprint arXiv:2610.12561.

[95] Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. DINO-WM: World Models on Pre-trained Visual Features Enable Zero-shot Planning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 79115–79135. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhou25t.html.

[96] Minjong Yoo, Jinwoo Jang, Sihyung Yoon, and Honguk Woo. World Model Implanting for Test-time Adaptation of Embodied Agents. In *International Conference on Machine Learning*, pages 72556–72573, 2025. URL https://proceedings.mlr.press/v267/yoo25a.html.

[97] 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*, page 9339, 2019. URL https://openaccess.thecvf.com/content_ICCV_2019/html/Savva_Habitat_A_Platform_for_Embodied_AI_Research_ICCV_2019_paper.html.

[98] Haoran Lu, Songling Liu, Yue Chen, Guo Ye, Mutian Shen, Shuyang Yu, Yu Xiao, Shang Wu, Jiayi Wang, Jianshu Zhang, Jihai Zhao, Xiangtian Gui, Chuye Hong, Yuran Wang, Maojiang Su, Ruihai Wu, Zhaoran Wang, and Han Liu. MagicSim: A Unified Infrastructure for Executable Embodied Interaction. *arXiv preprint arXiv:2606.17511v2*, 2026. URL https://arxiv.org/abs/2606.17511v2.

[99] Sherry Yang, Yilun Du, Seyed Ghasemipour, Jonathan Tompson, Leslie Kaelbling, Dale Schuurmans, and Pieter Abbeel. Learning Interactive Real-World Simulators. *International Conference on Learning Representations*, 2024:45210, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/c4d66eae503694424123b93ac0fbaf17-Abstract-Conference.html.

[100] Gemini Robotics Team, Krzysztof Choromanski, Coline Devin, Yilun Du, Debidatta Dwibedi, Ruiqi Gao, Abhishek Jindal, Thomas Kipf, Sean Kirmani, Isabel Leal, Fangchen Liu, Anirudha Majumdar, Andrew Marmon, Carolina Parada, Yulia Rubanova, Dhruv Shah, Vikas Sindhwani, Jie Tan, Fei Xia, Ted Xiao, Sherry Yang, Wenhao Yu, and Allan Zhou. Evaluating Gemini Robotics Policies in a Veo World Simulator. *arXiv preprint arXiv:2512.10675*, 2025. URL https://arxiv.org/abs/2512.10675.

[101] Xin Li, Bin He, Zhipeng Wang, Yanmin Zhou, Gang Li, and Zhongpan Zhu. A digital twin system for Task-Replanning and Human-Robot control of robot manipulation. *Advanced Engineering Informatics*, 62: 102570, 2024. doi: 10.1016/j.aei.2024.102570. URL https://doi.org/10.1016/j.aei.2024.102570.

[102] Youtian Lin, Yikang Yang, Zhanpeng Hu, Mengqi Zhou, Feihu Zhang, Xun Cao, Jiaheng Liu, and Yao Yao. Procedura: Agentic 3D Modeling with Procedural Control. *arXiv preprint arXiv:2608.26238*, 2026. URL https://arxiv.org/abs/2608.26238.

[103] Yichen Cai, Paul Jansonnie, Cristiana de Farias, Oleg Arenz, and Jan Peters. GaussTwin: Unified Simulation and Correction with Gaussian Splatting for Robotic Digital Twins. *arXiv preprint arXiv:2603.05108*, 2026. URL https://arxiv.org/abs/2603.05108.

[104] Jiayu Wang, Yue Yu, Bin Zhu, Zhiyao Yang, and Jingjing Chen. SpatialHarness: Test-Time Spatial Scaffolding for Fine Robotic Manipulation, 2026. URL https://arxiv.org/abs/2610.12457. arXiv preprint arXiv:2610.12457.

[105] Chengcheng Wang, Yuheng Shi, Siyu Xu, Yutong Wang, Jinjing Zhao, Chenghao Xia, Jinqi Liao, Jiankun Peng, Ligong Bi, Yuechuan Ma, Jiaqi Liu, Wenbo Wang, Zijian Wang, Yuemin Wu, Minjing Dong, Xiu Su, Tao Huang, Jianyuan Guo, Chengbin Du, Yunke Wang, and Chang Xu. Auriga Technical Report: A Task-Agnostic Agent Harness for Zero-Shot Embodied Manipulation, 2026. URL https://arxiv.org/abs/2611.01112. arXiv preprint arXiv:2611.01112.

[106] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551, 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

[107] 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*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

[108] Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi "Jim" Fan. CaP-X: A Framework for Benchmarking and Improving Coding Agents for Robot Manipulation. *arXiv preprint arXiv:2603.22435*, 2026. URL https://arxiv.org/abs/2603.22435.

[109] Mengzhao Jia, Yang Lin, Xixin Zhang, Zhihan Zhang, Xiaobai Liu, and Meng Jiang. Agent as Policy for Robotic Manipulation. *arXiv preprint arXiv:2609.12541*, 2026. URL https://arxiv.org/abs/2609.12541.

[110] Aidan Curtis, Nishanth Kumar, Jing Cao, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Trust the PRoC3S: Solving Long-Horizon Robotics Problems with LLMs and Constraint Satisfaction. In *Conference on Robot Learning*, pages 1362–1383, 2025. URL https://proceedings.mlr.press/v270/curtis25a.html.

[111] Denis Shcherba, Eckart Cobo-Briesewitz, Cornelius V. Braun, and Marc Toussaint. Meta-Optimization and Program Search using Language Models for Task and Motion Planning. In *Conference on Robot Learning*, pages 5339–5361, 2025. URL https://proceedings.mlr.press/v305/shcherba25a.html.

[112] Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning. In *Conference on Robot Learning*, pages 23–72, 2023. URL https://proceedings.mlr.press/v229/rana23a.html.

[113] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/28e50ee5b72e90b50e7196fde8ea260e-Abstract-Conference.html.

[114] Mingxin Li, Enge Song, Yueshang Zuo, Xiaodong Liu, Rong Wen, Qiang Fu, Gianni Antichi, Jian He, Jing Tie, Zhou Shao, Xiaobo Xue, Xiong Xiao, Luyao Zhong, Shaokai Zhang, Jiangu Zhao, Jianyuan Lu, Shize Zhang, Xiaoqing Sun, Changgang Zheng, Zihao Fan, Haonan Li, Tian Pan, Xiaomin Wu, Yang Song, Xing Li, Biao Lyu, Meng Li, Haipeng Dai, Guihai Chen, and Shunmin Zhu. Scalable LLM Agent Tool Access in the Cloud. *arXiv preprint arXiv:2607.15593v1*, 2026. doi: 10.48550/arXiv.2607.15593. URL https://arxiv.org/abs/2607.15593v1.

[115] Liangtao Lin, Qingang Zhang, Zhaomeng Zhu, Tianwei Zhang, and Yonggang Wen. Task-Aware Harness Provisioning for LLM Agents in Mission-Critical Infrastructure Operations. *arXiv preprint arXiv:2608.17433*, 2026. URL https://arxiv.org/abs/2608.17433.

[116] Jinbang Huang, Yuanzhao Hu, Zhiyuan Li, Ran Qi, Yixin Xiao, Zhanguang Zhang, Mark Coates, Tongtong Cao, and Yingxue Zhang. RoboHarness: Memory-Driven Orchestration of Heterogeneous Robot Policies for Long-Horizon Planning. *arXiv preprint arXiv:2607.18060*, 2026. URL https://arxiv.org/abs/2607.18060.

[117] Ajay Kumar Tanwani, Raghav Anand, Joseph E. Gonzalez, and Ken Goldberg. RILaaS: Robot Inference and Learning as a Service. *IEEE Robotics and Automation Letters*, 5(3):4423–4430, 2020. doi: 10.1109/lra.2020.2998414. URL https://doi.org/10.1109/lra.2020.2998414.

[118] Jeffrey Ichnowski, Kaiyuan Chen, Karthik Dharmarajan, Simeon Adebola, Michael Danielczuk, Víctor Mayoral-Vilches, Nikhil Jha, Hugo Zhan, Edith LLontop, Derek Xu, Camilo Buscaron, John Kubiatowicz, Ion Stoica, Joseph Gonzalez, and Ken Goldberg. FogROS2: An Adaptive Platform for Cloud and Fog Robotics Using ROS 2. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5493–5500, 2023. doi: 10.1109/ICRA48891.2023.10161307. URL https://doi.org/10.1109/ICRA48891.2023.10161307.

[119] Zihao Zheng, Hangyu Cao, Jiayu Chen, Sicheng Tian, Chenyue Li, Maoliang Li, Xinhao Sun, Guojie Luo, and Xiang Chen. RoboECC: Multi-Factor-Aware Edge-Cloud Collaborative Deployment for VLA Models. *arXiv preprint arXiv:2603.20711*, 2026. URL https://arxiv.org/abs/2603.20711.

[120] Anna Li, Christina Giannoula, and Nandita Vijaykumar. KerColle: Unlocking Fine-Grained GPU Concurrency in Vision-Language-Action Models. *arXiv preprint arXiv:2609.22335*, 2026. URL https://arxiv.org/abs/2609.22335.

[121] Yinwei Dai, Ganesh Ananthanarayanan, Landon Cox, Xenofon Foukas, Bozidar Radunovic, and Ravi Netravali. Kairos: A Scalable Serving System for Physical AI. *arXiv preprint arXiv:2605.11381*, 2026. URL https://arxiv.org/abs/2605.11381.

[122] Chenghua Wang, Daliang Xu, Dongqi Cai, Duojin Sun, Hao Zhang, Haoze Qian, Huaiyuan Zhang, Jinshuo Cui, Junbo Cui, Kezhao Zhao, Longxi Gao, Mengwei Xu, Rongjie Yi, Ruixin Liu, Shangguang Wang, Tam Sikyuen, Tianyue Zhang, Weikai Xie, Xuanzhe Liu, Yingying Qin, Yiwen Lu, Yuan Yao, Yuezhi Zu, Yunhan Guo, Yuxin Zheng, and Ziqi Guo. PhyAI: Real-Time Physical AI at the Edge, Scalable Rollouts in the Cloud. *arXiv preprint arXiv:2608.03682*, 2026. URL https://arxiv.org/abs/2608.03682.

[123] Zhiyuan Xu, Kun Wu, Junjie Wen, Jinming Li, Ning Liu, Zhengping Che, and Jian Tang. A Survey on Robotics with Foundation Models: toward Embodied AI. *arXiv preprint arXiv:2402.02385*, 2024. doi: 10.48550/arXiv.2402.02385. URL https://arxiv.org/abs/2402.02385.

[124] 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 *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530, 2023. doi: 10.1109/ICRA48891.2023.10161317. URL https://doi.org/10.1109/ICRA48891.2023.10161317.

[125] Hao-Shu Fang, Chenxi Wang, Hongjie Fang, Minghao Gou, Jirong Liu, Hengxu Yan, Wenhai Liu, Yichen Xie, and Cewu Lu. AnyGrasp: Robust and Efficient Grasp Perception in Spatial and Temporal Domains. *IEEE Transactions on Robotics*, 39(5):3929–3945, 2023. doi: 10.1109/TRO.2023.3281153. URL https://doi.org/10.1109/TRO.2023.3281153.

[126] Krishna Jatavallabhula, Alihusein Kuwajerwala, Qiao Gu, Mohd Omama, Ganesh Iyer, Soroush Saryazdi, Tao Chen, Alaa Maalouf, Shuang Li, Nikhil Keetha, Ayush Tewari, Joshua Tenenbaum, Celso Melo, Madhava Krishna, Liam Paull, Florian Shkurti, and Antonio Torralba. ConceptFusion: Open-set multimodal 3D mapping. In *Robotics: Science and Systems XIX*, 2023. doi: 10.15607/rss.2023.xix.066. URL https://doi.org/10.15607/rss.2023.xix.066.

[127] Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual Language Maps for Robot Navigation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10608–10615, 2023. doi: 10.1109/ICRA48891.2023.10160969. URL https://doi.org/10.1109/ICRA48891.2023.10160969.

[128] Peiqi Liu, Yaswanth Orru, Jay Vakil, Chris Paxton, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Demonstrating OK-Robot: What Really Matters in Integrating Open-Knowledge Models for Robotics. In *Robotics: Science and Systems XX*. Robotics: Science and Systems Foundation, 2024. doi: 10.15607/RSS.2024.XX.091. URL https://www.roboticsproceedings.org/rss20/p091.html.

[129] Zexi Li, Yehang Zhang, Wenqian Li, Haojian Huang, Chenxu Wang, Shiyuan Deng, Yangkai Wei, Tianyi Zhang, Binghui Xie, Bohan Zhou, Yifan Chang, Kaiwen Zhou, Ying-Cong Chen, James Cheng, and Yinchuan Li. Robo-Harness K1: Harnessing Robot-Use Agents via Perception Augmentation. *arXiv preprint arXiv:2609.29389*, 2026. URL https://arxiv.org/abs/2609.29389.

[130] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models. In *Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 540–562. PMLR, 2023. URL https://proceedings.mlr.press/v229/huang23b.html.

[131] Dhruv Shah, Ajay Sridhar, Nitish Dashora, Kyle Stachowicz, Kevin Black, Noriaki Hirose, and Sergey Levine. ViNT: A Foundation Model for Visual Navigation. In *Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 711–733. PMLR, 2023. URL https://proceedings.mlr.press/v229/shah23a.html.

[132] Hengyuan Hu, Priya Sundaresan, Jensen Gao, and Dorsa Sadigh. VIA: Visual Interface Agent for Robot Control. *arXiv preprint arXiv:2607.11119*, 2026. URL https://arxiv.org/abs/2607.11119.

[133] Yehang Zhang, Haojian Huang, Yifan Chang, Jianchong Su, Bohan Zhou, Yingjie Xu, Wosong Chen, Tianhao Zhou, Chenxu Wang, Tianyi Zhang, Yangkai Wei, Wenqian Li, Shiyuan Deng, Yinchuan Li, Ying-Cong Chen, and Zexi Li. World Action Agent: Harnessing VLMs for Robot Manipulation via World Action Rehearsal. *arXiv preprint arXiv:2609.29964*, 2026. URL https://arxiv.org/abs/2609.29964.

[134] Petter Ögren and Christopher I. Sprague. Behavior Trees in Robot Control Systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 5:81–107, 2022. doi: 10.1146/annurev-control-042920-095314. URL https://doi.org/10.1146/annurev-control-042920-095314.

[135] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Robotics: Science and Systems XIX*, 2023. doi: 10.15607/RSS.2023.XIX.016. URL https://doi.org/10.15607/RSS.2023.XIX.016.

[136] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Robotics: Science and Systems XIX*, 2023. doi: 10.15607/rss.2023.xix.026. URL https://doi.org/10.15607/rss.2023.xix.026.

[137] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: Robotics Transformer for Real-World Control at Scale. *arXiv preprint arXiv:2212.06817*, 2022. URL https://arxiv.org/abs/2212.06817.

[138] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An Open-Source Generalist Robot Policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024. doi: 10.15607/RSS.2024.XX.090. URL https://doi.org/10.15607/RSS.2024.XX.090.

[139] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization. In *Conference on Robot Learning*, pages 17–40, 2025. URL https://proceedings.mlr.press/v305/black25a.html.

[140] Yutong Hu, Fengjiao Chen, Xuezhi Cao, and Renaud Detry. 2AM: Grounding Agent-Side Memory as Guidance for Steerable Action Models in Long-Horizon Manipulation. *arXiv preprint arXiv:2609.11308*, 2026. URL https://arxiv.org/abs/2609.11308.

[141] Yicheng Jiang, Zesen Gan, Xiaobo Wang, Tianlun He, Chenxu Zhao, Minghui Wu, Xinyue Wang, Jiaxu Wang, Junhao He, Jianan Wang, and Qiming Shao. AR-WAM: A Visual-Conditioned Agent-Ready World Action Model for Robotic Manipulation. *arXiv preprint arXiv:2609.23578*, 2026. URL https://arxiv.org/abs/2609.23578.

[142] Yilun Du, Sherry Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning Universal Policies via Text-Guided Video Generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0403. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1d5b9233ad716a43be5c0d3023cb82d0-Abstract-Conference.html.

[143] Chuning Zhu, Raymond Yu, Siyuan Feng, Benjamin Burchfiel, Paarth Shah, and Abhishek Gupta. Unified World Models: Coupling Video and Action Diffusion for Pretraining on Large Robotic Datasets. In *Robotics: Science and Systems XXI*, 2025. doi: 10.15607/rss.2025.xxi.015. URL https://doi.org/10.15607/rss.2025.xxi.015.

[144] Chi-Lam Cheang, Guangzeng Chen, Ya Jing, Tao Kong, Hang Li, Yifeng Li, Yuxiao Liu, Hongtao Wu, Jiafeng Xu, Yichu Yang, Hanbo Zhang, and Minzhao Zhu. GR-2: A Generative Video-Language-Action Model with Web-Scale Knowledge for Robot Manipulation. *arXiv preprint arXiv:2410.06158*, 2024. doi: 10.48550/arXiv.2410.06158. URL https://arxiv.org/abs/2410.06158.

[145] Shuang Li, Yihuai Gao, Dorsa Sadigh, and Shuran Song. Unified Video Action Model. *arXiv preprint arXiv:2503.00200*, 2025. doi: 10.48550/arXiv.2503.00200. URL https://arxiv.org/abs/2503.00200.

[146] Siyin Wang, Junhao Shi, Zhaoyang Fu, Xinzhe He, Feihong Liu, Chenchen Yang, Yikang Zhou, Zhaoye Fei, Jingjing Gong, Jinlan Fu, Mike Zheng Shou, Xuanjing Huang, Xipeng Qiu, and Yu-Gang Jiang. World Action Models: The Next Frontier in Embodied AI. *arXiv preprint arXiv:2605.12090*, 2026. doi: 10.48550/arXiv.2605.12090. URL https://arxiv.org/abs/2605.12090.

[147] Muhammad Raza Naqvi, Arkopaul Sarkar, Farhad Ameri, Linda Elmhadhbi, Thierry Louge, and Mohamed Hedi Karray. Ontology-driven integration of advertised and operational capabilities in robots. *Scientific Reports*, 15:34326, 2025. doi: 10.1038/s41598-025-16649-3. URL https://doi.org/10.1038/s41598-025-16649-3.

[148] Xue Qin, Simin Luan, Cong Yang, and Zhijun Li. ECM Contracts: Contract-Aware, Versioned, and Governable Capability Interfaces for Embodied Agents. *arXiv preprint arXiv:2604.13097*, 2026. URL https://arxiv.org/abs/2604.13097.

[149] Qingzhou Lu, Chenxin Liu, Xinming Wang, Liu Cao, Yanjiang Guo, Mengdi Xu, and Jianyu Chen. HITCH: Teaching a Vision-Language Model to Operate a Humanoid through a Whole-Body Harness, 2026. URL https://arxiv.org/abs/2611.08039. arXiv preprint arXiv:2611.08039.

[150] Chao Liang, Ke Chen, Fengxiaoxiao Li, Shibo Huang, Yufeng Sun, Zhaokai Yin, Jianhuan Lai, Haozhe Feng, Chenghuan Jia, and Zhipeng Zhang. Toward the Era of Agentic Embodied Intelligence, 2026. URL https://arxiv.org/abs/2611.09124. arXiv preprint arXiv:2611.09124.

[151] Kehong Gong, Xiaoxu Zheng, Guanli Hou, Seunghun Lee, Mengqi Guo, Runjia Tan, Zhengyu Wen, Qi Wang, Xiaoyu He, Mingyuan Zhang, and Michael Bi Mi. TinyAstral: Transferring Self-Evolving Harnesses to Tiny Generalist Manipulation Agents, 2026. URL https://arxiv.org/abs/2610.41216. arXiv preprint arXiv:2610.41216.

[152] Xue Qin, Simin Luan, John See, Cong Yang, and Zhijun Li. AEROS: A Single-Agent Operating Architecture with Embodied Capability Modules. *arXiv preprint arXiv:2604.07039*, 2026. URL https://arxiv.org/abs/2604.07039.

[153] 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 "Jim" Fan, and Guanzhi Wang. ASPIRE: Agentic /Skills Discovery for Robotics. *arXiv preprint arXiv:2607.00272*, 2026. URL https://arxiv.org/abs/2607.00272.

[154] Ruofei Ju, Xinrui Wang, Xin Ding, Yifan Yang, Hao Wu, Shiqi Jiang, Qianxi Zhang, Hao Wen, Xiangyu Li, Weijun Wang, Kun Li, Yunxin Liu, Haipeng Dai, Wei Wang, and Ting Cao. Embodiskill: Skill-aware reflection for self-evolving embodied agents, 2026. URL https://arxiv.org/abs/2605.10332.

[155] Yitong Chen, Zezheng Huai, Sixian Li, Yubang Wang, Haozhe Zhang, Yifei Zhang, Hechang Chen, Jingjing Gong, Yu-Gang Jiang, and Xipeng Qiu. ETA: A New Agentic Paradigm for Embodied Tasks. *arXiv preprint arXiv:2608.03924*, 2026. URL https://arxiv.org/abs/2608.03924.

[156] Irvin Steve Cardenas, Marcus Anthony Arnett, Natalie Catherine Yeo, Lucky Sah, and Jong-Hoon Kim. ROSClaw: An OpenClaw ROS 2 Framework for Agentic Robot Control and Interaction. *arXiv preprint arXiv:2603.26997*, 2026. URL https://arxiv.org/abs/2603.26997.

[157] Jingpu Yang, Fengxian Ji, Zhengzhao Lai, Zhexuan Cui, Guangxian Ouyang, Qian Jiang, Fan Zhang, Min Peng, Qianqian Xie, Preslav Nakov, and Zhuohan Xie. LabGuard: Grounding Natural-Language Laboratory Rules into Runtime Guards for Embodied Laboratory Agents. *arXiv preprint arXiv:2606.31045*, 2026. URL https://arxiv.org/abs/2606.31045.

[158] Xue Qin, Simin Luan, John See, Zeyd Boukhers, Cong Yang, and Zhijun Li. Harnessing Embodied Agents: Runtime Governance for Policy-Constrained Execution. *arXiv preprint arXiv:2604.07833*, 2026. URL https://arxiv.org/abs/2604.07833.

[159] Zixuan Wang, Yuqi Liu, Yuxin Chen, Sengqiao Yang, Jinhui Ye, Pengguang Chen, Changsheng Lu, Shu Liu, Bei Yu, and Jiaya Jia. CarryOn: A Dual-Loop Harness for Personalized and Reliable Robot Assistance, 2026. URL https://arxiv.org/abs/2610.21453. arXiv preprint arXiv:2610.21453.

[160] Zixiang Zhao, Yansong Feng, Yang Yang, Chaoyu Wang, Haoran Xiao, Hui Zhang, Chuang Cheng, and Jianjun Ma. CommitFlow: Semantic Commitment Verification and Local Correction for Long-Horizon Robot Manipulation VLA Execution. *arXiv preprint arXiv:2609.21908*, 2026. URL https://arxiv.org/abs/2609.21908.

[161] Wenze Wang, Mehdi Hosseinzadeh, and Feras Dayoub. A Physical Agentic Loop for Language-Guided Grasping with Execution-State Monitoring. *arXiv preprint arXiv:2604.07395*, 2026. URL https://arxiv.org/abs/2604.07395.

[162] Fabio Hübel, Adam Lilja, Yue Yin, Siming He, Claire Tomlin, Shankar Sastry, Jonas Frey, and Marco Pavone. BT-Agent: Deploying Onboard Robot Agents with Behavior Trees and Multi-Fidelity Evaluation, 2026. URL https://arxiv.org/abs/2610.13862. arXiv preprint arXiv:2610.13862.

[163] Tao Lin, Yuxin Du, Haoxiao Wang, Zile Zhong, and Bo Zhao. FastHarness: Experience-Driven Review Scheduling for Vision-Language-Action Policies, 2026. URL https://arxiv.org/abs/2610.31427. arXiv preprint arXiv:2610.31427.

[164] Lijian Lin, Ye Zhu, Fan Zhang, Yunfei Liu, Baofeng Li, Xianwen Zeng, Jianan Wang, and Yu Li. MIRA: Real-Time Full-Duplex Human-Robot Interaction for Embodied Companions. *arXiv preprint arXiv:2609.24547*, 2026. URL https://arxiv.org/abs/2609.24547.

[165] Kailin Wang, Haoxiang Jie, Yaoyuan Yan, Zhiyou Heng, and Zhaosong Li. RegenHarness: A Robot Agent Harness with Evidence-Gated Recursive Self-Improvement. *arXiv preprint arXiv:2609.27612*, 2026. URL https://arxiv.org/abs/2609.27612.

[166] Shengbang Liu, Zhengye Du, Zhilong Wan, Honghao Su, Chenxiang Xia, Chang Ge, Jinyang Xiao, Nan Wang, Zhipeng Hu, Huining Yuan, Shu'ang Yu, Yue Chen, Tianxing Chen, Chuankang Li, Maoqing Yao, Wenbo Ding, Yu Wang, Guanghui Ren, and Chao Yu. RoboValue: A Fine-Grained Sim-and-Real Benchmark for Unified Evaluation of Robotic Value Models, 2026. URL https://arxiv.org/abs/2610.26663. arXiv preprint arXiv:2610.26663.

[167] Yang Wan, Xihang Yue, Zhirui Liu, Ziyuan Chu, Shuxun Wang, Yuhan Chen, Xiaonan Jiang, Xukun Zhu, Yubo Dong, and Linchao Zhu. No Free Checker: A Survey of Verifiers for Robot Policies. *arXiv preprint arXiv:2609.09250*, 2026. URL https://arxiv.org/abs/2609.09250.

[168] Lambert Lee and Joshua Lau. Robot Context Protocol (RCP): A Runtime-Agnostic Interface for Agent-Aware Robot Control. *arXiv preprint arXiv:2506.11650*, 2025. URL https://arxiv.org/abs/2506.11650.

[169] Quilee Simeon, Justin M. Wei, and Yile Fan. Octopus Protocol: One-Shot Hardware Discovery and Control for AI Agents via Infrastructure-as-Prompts. *arXiv preprint arXiv:2605.09055*, 2026. URL https://arxiv.org/abs/2605.09055.

[170] Junhao Shi, Zezheng Huai, Siyin Wang, Jia Chen, Yubang Wang, Zhaoye Fei, Hechang Chen, Jingjing Gong, Xipeng Qiu, and Yu-Gang Jiang. Advancing Omnimodal Embodied Agents from Isolated Skills to Everyday Physical Autonomy. *arXiv preprint arXiv:2606.27251*, 2026. URL https://arxiv.org/abs/2606.27251.

[171] Heqing Yang, Yang Yi, Liyao Wang, Linqing Zhong, Donglin Yang, Ruipu Wu, Zitong Bai, Fengjiao Chen, Manyuan Zhang, Linjiang Huang, and Si Liu. EVA-Client: A Unified Data Collection, Inference, and Deployment Framework for Embodied Policies on Real Robots. *arXiv preprint arXiv:2607.02646*, 2026. URL https://arxiv.org/abs/2607.02646.

[172] Pablo Ortega-Kral, Eliot Xing, Arthur Bucker, Vernon Luk, Junseo Kim, Owen Kwon, Angchen Xie, Nikhil Sobanbabu, Yifu Yuan, Megan Lee, Deepam Ameria, Bhaswanth Ayapilla, Jaycie Bussell, Guanya Shi, Jonathan Francis, and Jean Oh. RIO: Flexible Real-Time Robot I/O for Cross-Embodiment Robot Learning. *arXiv preprint arXiv:2605.11564*, 2026. URL https://arxiv.org/abs/2605.11564.

[173] Arturo Laurenzi, Davide Antonucci, Nikos G. Tsagarakis, and Luca Muratore. The XBot2 real-time middleware for robotics. *Robotics and Autonomous Systems*, 163:104379, 2023. doi: 10.1016/j.robot.2023.104379. URL https://doi.org/10.1016/j.robot.2023.104379.

[174] Daniel Casini, Jian-Jia Chen, Jing Li, Federico Reghenzani, and Harun Teper. A Survey of Real-Time Support, Analysis, and Advancements in ROS 2. *Leibniz Transactions on Embedded Systems*, 11(1):1:1–1:37, 2026. doi: 10.4230/LITES.11.1.1. URL https://drops.dagstuhl.de/entities/document/10.4230/LITES.11.1.1.

[175] Bastien Dussard and Guillaume Sarthou. Extracting Semantics: LLM-Guided Automatic Population of Robot Ontology from URDF. *arXiv preprint arXiv:2606.17073*, 2026. URL https://arxiv.org/abs/2606.17073.

[176] Kajetan Rachwał, Maciej Majek, Bartłomiej Boczek, Kacper Dąbrowski, Paweł Liberadzki, Adam Dąbrowski, and Maria Ganzha. RAI: Flexible Agent Framework for Embodied AI. *arXiv preprint arXiv:2505.07532*, 2025. URL https://arxiv.org/abs/2505.07532.

[177] Xiaolin Zhou, Liu Liu, Tingyang Xiao, Wei Feng, Fa Fu, Xinrui Meng, Xinjie Wang, Jialiang Han, Boyang Yu, Yun Du, Wei Sui, and Zhizhong Su. HoloAgent-0: A Unified Embodied Agent Framework with 3D Spatial Memory. *arXiv preprint arXiv:2606.23565*, 2026. URL https://arxiv.org/abs/2606.23565.

[178] Xue Qin, Simin Luan, Cong Yang, and Zhijun Li. Federated Trust for Embodied Robot Capability Marketplaces. *arXiv preprint arXiv:2609.00404*, 2026. URL https://arxiv.org/abs/2609.00404.

[179] Weifan Guan, Qinghao Hu, Huasen Xi, Chenxiao Zhang, Aosheng Li, and Jian Cheng. RoboNeuron: A Middle-Layer Infrastructure for Agent-Driven Orchestration in Embodied AI. *arXiv preprint arXiv:2512.10394*, 2025. URL https://arxiv.org/abs/2512.10394.

[180] Dongxu Yang. Device Context Protocol: A Compact, Safety-First Architecture for LLM-Driven Control of Constrained Devices. *arXiv preprint arXiv:2605.26159*, 2026. URL https://arxiv.org/abs/2605.26159.

[181] Jianwei Tai. Same Weights, Different Robot: A Deployment Safety View of VLA Policies. *arXiv preprint arXiv:2606.03724*, 2026. URL https://arxiv.org/abs/2606.03724.

[182] Tian Nian, Weijie Ke, Shaolong Zhu, and Bingshan Hu. Control Your Robot: A Unified System for Robot Control and Policy Deployment. *arXiv preprint arXiv:2509.23823*, 2025. URL https://arxiv.org/abs/2509.23823.

[183] Ling Xu, Borui Li, Hao Wu, Chuyu Han, Xiangyu Li, Mohan Hua, Shiqi Jiang, Ting Cao, Chuanyou Li, Sheng Zhong, and Shuai Wang. Embodied.cpp: A Portable Inference Runtime of Embodied AI Models on Heterogeneous Robots. *arXiv preprint arXiv:2607.02501*, 2026. URL https://arxiv.org/abs/2607.02501.

[184] Lukas Lürzer, Hannes Unger, and Stefan Huber. Reconstructing OPC UA Address Spaces from Time-Series Databases. *arXiv preprint arXiv:2606.10663*, 2026. URL https://arxiv.org/abs/2606.10663.

[185] Fuyu Ma, Dong Li, Yizhe Chen, Jiahui Xing, Yu Liu, Dapeng Lan, Jinyan Shao, and Zhibo Pang. Orchestrating Embodied Systems through the Embodied Context Protocol: Motivation, Progress, and Directions. *Research*, 8, 2025. doi: 10.34133/research.1047. URL https://doi.org/10.34133/research.1047.

[186] Yongsheng Zhao, Lei Zhao, Baoping Cheng, Gongxin Yao, Xuanzhang Wen, and Han Gao. VLA-RAIL: A Real-Time Asynchronous Inference Linker for VLA Models and Robots. *arXiv preprint arXiv:2512.24673*, 2025. URL https://arxiv.org/abs/2512.24673.

[187] Jiarui Yuan, Yuxin Zuo, Xiaoyu Huang, Wenbo Zhang, Kwanyoung Park, Junlin Yang, Tianxiang Zhang, Ruoqu Chen, Yutao Ouyang, Bingxiang He, Xingtai Lv, Hongjin Wu, Jiajun Liu, Jiahao Zhao, Tianwei Luo, Haozhan Li, Cheng Qian, Jiaxin Ge, Huan-ang Gao, Kaisen Yang, Jiaqi Deng, Qixin Xu, Siqiao Huang, Bowen Zhou, Mengdi Xu, Zhiyuan Liu, Pieter Abbeel, and Ning Ding. Democratizing Robot-Use Agents: A Unified Harness with Open Models, 2026. URL https://arxiv.org/abs/2611.01560. arXiv preprint arXiv:2611.01560.

[188] Junwu Xiong, Jiaxuan Gao, Wei Chai, Renxing Chen, Yuzhen Li, Yu Guo, Yucheng Guo, Mingxi Luo, Wenyang Ma, Yiyun Mou, Yifei Zhang, Chen Zhou, and Yongjian Guo. Embodied Operators and Benchmarking: Toward Reusable and Deployable Embodied Intelligence Systems. *arXiv preprint arXiv:2607.03283*, 2026. URL https://arxiv.org/abs/2607.03283.

[189] Renan Souza, Amal Gueroudji, Stephen DeWitt, Daniel Rosendo, Tirthankar Ghosal, Robert Ross, Prasanna Balaprakash, and Rafael Ferreira da Silva. PROV-AGENT: Unified Provenance for Tracking AI Agent Interactions in Agentic Workflows. *arXiv preprint arXiv:2508.02866*, 2025. URL https://arxiv.org/abs/2508.02866.

[190] Yiqi Wang, Jiaqi Zhang, Zhangkai Wu, Taotao Cai, Zirui Liu, Zequn Sun, Manqing Dong, Mingkai Zheng, Yiqun Duan, Xuefei Yin, and Yanming Zhu. From Agent Traces to Trust: A Survey of Evidence Tracing and Execution Provenance in LLM Agents. *arXiv preprint arXiv:2606.04990v5*, 2026. URL https://arxiv.org/abs/2606.04990v5.

[191] Yuezhang Lv, Yunzhou Zhang, Chao Lu, Jiajun Zhu, and Song Wu. Targetless Intrinsics and Extrinsic Calibration of Multiple LiDARs and Cameras with IMU using Continuous-Time Estimation. *arXiv preprint arXiv:2501.02821*, 2025. URL https://arxiv.org/abs/2501.02821.

[192] Zhiwei Huang, Jiaqi Li, Hongbo Zhao, Xiao Ma, Ping Zhong, Xiao-Hu Zhou, Wei Ye, and Rui Fan. Environment-Driven Online LiDAR-Camera Extrinsic Calibration. *IEEE Transactions on Automation Science and Engineering*, 22:24164–24176, 2025. doi: 10.1109/TASE.2025.3627253. URL https://doi.org/10.1109/TASE.2025.3627253.

[193] Zeyu Lou, Yanhong Zeng, Yong Wang, and Chenyang Si. Know Your Body: A Harness for Direct and Self-Improving Robot Control with VLMs. *arXiv preprint arXiv:2609.28530*, 2026. URL https://arxiv.org/abs/2609.28530.

[194] Felipe H. N. Nascimento, Sabrina A. Cardoso, Antonio M. N. Lima, and Danilo F. S. Santos. Synchronizing a collaborative arm's digital twin in real-time. In *2023 Latin American Robotics Symposium (LARS), 2023 Brazilian Symposium on Robotics (SBR), and 2023 Workshop on Robotics in Education (WRE)*, pages 230–235, 2023. doi: 10.1109/lars/sbr/wre59448.2023.10333055. URL https://doi.org/10.1109/lars/sbr/wre59448.2023.10333055.

[195] Harun Teper, Shaokai Lin, Shulu Li, Edward A. Lee, and Jian-Jia Chen. Deterministic Execution of ROS 2 Applications via Lingua Franca. *arXiv preprint arXiv:2606.09203*, 2026. URL https://arxiv.org/abs/2606.09203.

[196] Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. Vlash: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025. URL https://arxiv.org/abs/2512.01031.

[197] Yangqing Zheng, Shunqi Mao, Dingxin Zhang, and Weidong Cai. Reflex First, Reflect Later: Latency-Aware Embodied LLM Agents for Dynamic Response. *arXiv preprint arXiv:2506.07223*, 2025. URL https://arxiv.org/abs/2506.07223.

[198] Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022, 2018. doi: 10.1016/j.ifacol.2018.08.474. URL https://doi.org/10.1016/j.ifacol.2018.08.474.

[199] Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian "Max" Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, Jimmy Wu, Guanzhi Wang, S. Shankar Sastry, Ken Goldberg, Linxi "Jim" Fan, Yuke Zhu, and Guanya Shi. ENPIRE: Agentic Robot Policy Self-Improvement in the Real World. *arXiv preprint arXiv:2606.19980*, 2026. URL https://arxiv.org/abs/2606.19980.

[200] Fu Chen, Xin Ding, Bingjia Huang, Xiangyu Li, Mingju Wang, Jiawei He, Kun Li, Wei Sun, Yunxin Liu, Hao Wu, and Ting Cao. Zeva: In-context causal learning for generalizable embodied manipulation, 2026. URL https://arxiv.org/abs/2608.30880.

[201] Dai Zhou, Jiexi Yan, Tong Li, Yuxuan Wang, and Cheng Deng. Remember smarter: Visual history compressor and hyperbolic experience space for robotic memory, 2026. URL https://arxiv.org/abs/2608.15269.

[202] Hongbo Gao, Zeyu Ni, Xin Wen, Siyu Xu, and Ruifeng Li. Agm: Achievement-grounded memory for closed-loop agents with frozen vla policies, 2026. URL https://arxiv.org/abs/2608.29537.

[203] Chang Nie, Zhe Liu, and Hesheng Wang. Teach and grow: An agent-centered architecture for general robot learning, 2026. URL https://arxiv.org/abs/2608.17209.

[204] Riyaaz Shaik and Chandru Venkataraman. Refactor-vla: Unsupervised library learning of typed motor programs, 2026. URL https://arxiv.org/abs/2609.01215.

[205] Xue Qin, Simin Luan, Cong Yang, and Zhijun Li. Evolving Skill Modules under a Fixed Planner: Versioning, Rollback, and Runtime Governance for Long-Lived Robot Systems. *arXiv preprint arXiv:2604.07799v3*, 2026. URL https://arxiv.org/abs/2604.07799v3.

[206] Xue Qin, Simin Luan, John See, Zeyd Boukhers, Cong Yang, and Zhijun Li. ICAN-Deploy: Identity-Stable Canary Deployment for Safety-Critical Embodied Agents. *arXiv preprint arXiv:2605.28097*, 2026. URL https://arxiv.org/abs/2605.28097.

[207] Hang Yu. You don't need to stay in the loop: An agentic robotics loop for robot-policy improvement, 2026. URL https://arxiv.org/abs/2608.07555.

[208] Zhongxi Chen and Shenqi Zong. Onevomemory: Evolving memory through online robot rollouts for pretrained robot policies, 2026. URL https://arxiv.org/abs/2608.08749.

[209] Jiayi Tian, Shiao Liu, Yuting Xu, Jia Lu, Zihao Guan, Honglin Han, Di Yang, Minqi Gu, Yifei Qian, Tianlin Zhang, Yanqing Zhu, Zeqian Ye, Menglin Yang, Fei Wang, Xu Hu, Xiuxian Li, Wei Zhang, Shihui Su, Yiyan Ji, Jingbo Wang, Ziteng Feng, Jiaheng Liu, Zhaoxiang Zhang, Xiaolong Wu, Zixiao Tang, Zhining Gu, Yang Cai, Linbo Zheng, Jingjing Ma, Mingyang Yin, Zedong Chu, Wenbin Tang, and Mu Xu. ABot-AgentOS: A General Robotic Agent OS with Lifelong Multi-modal Memory. *arXiv preprint arXiv:2607.10350*, 2026. URL https://arxiv.org/abs/2607.10350.

[210] Xianyao Li, Ruitong Tian, Rui Min, Fang Xu, and Eric Jing Du. DreamLedger: Where to Refuse World-Model Imagination Using Execution-Settled Credit, 2026. URL https://arxiv.org/abs/2608.23863v4.

[211] Chen Chen, Omid Bagherieh, Naveen Nikamanth Appiah Balaji, Jesse T. Gonzalez, Negin Heravi, Fahad Islam, Xueting Li, Jean-Baptiste Regli, Kejia Ren, Petr Švec, David J. Yoon, Matthias Müller, and Vladlen Koltun. On Agentic Manipulation Policies, 2026. URL https://arxiv.org/abs/2611.09892. arXiv preprint arXiv:2611.09892.

[212] Georgios Tziafas and Hamidreza Kasaei. Lifelong Robot Library Learning: Bootstrapping Composable and Generalizable Skills for Embodied Control with Language Models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 515–522, 2024. doi: 10.1109/ICRA57147.2024.10611448. URL https://doi.org/10.1109/ICRA57147.2024.10611448.

[213] Ziyi Bai, Siqi Li, Tinglei Huang, and Börje F. Karlsson. Practice: From experience to expertise in self-evolving embodied agents, 2026. URL https://arxiv.org/abs/2608.30760.

[214] Bowen Li, Mayank Mishra, Y. Isabel Liu, Stone Tao, Nishanth Kumar, Alexander G. Gray, Ruwan Wickramarachchi, Jonathan Francis, Sebastian Scherer, and Tom Silver. Recover, discover, plan: Learning skills and concepts from robot failures, 2026. URL https://arxiv.org/abs/2606.18328.

[215] Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi Gu, Xunliang Cai, Xiang Wang, and An Zhang. Skill1: Unified evolution of skill-augmented agents via reinforcement learning. *arXiv preprint arXiv:2605.06130*, 2026. doi: 10.48550/arXiv.2605.06130. URL https://arxiv.org/abs/2605.06130.

[216] Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026. doi: 10.48550/arXiv.2602.08234. URL https://arxiv.org/abs/2602.08234.

[217] Junyi Zhang, Jiaxin Ge, Hanjun Yoo, Letian Fu, Zihan Yang, Yaowei Liu, Raj Saravanan, Shaofeng Yin, Justin Yu, Dantong Niu, Zirui Wang, Roei Herzig, Ken Goldberg, Yutong Bai, David M. Chan, Ion Stoica, Angjoo Kanazawa, Jiahui Lei, Haiwen Feng, and Trevor Darrell. Playful Agentic Robot Learning. *arXiv preprint arXiv:2606.19419*, 2026. URL https://arxiv.org/abs/2606.19419.

[218] Seungyeon Kim, Junhoo Lee, Minkyu Kim, Baekseung Kim, and Nojun Kwak. Recursive Harness Distillation across Agents for Robot Manipulation, 2026. URL https://arxiv.org/abs/2609.33378. arXiv preprint arXiv:2609.33378.

[219] Shengguang Wu, Suning Huang, Cherie Ho, Ria Doshi, Hao Zhu, and Serena Yeung-Levy. RoboArchitect: Automating the Design of Robotic Agents with Persistent Memory, 2026. URL https://arxiv.org/abs/2610.42141. arXiv preprint arXiv:2610.42141.

[220] Xin Wang, Wenhao Wu, Menghao Zhang, Zhi Wang, Kun Shao, Jian Luan, Yang Li, Qing Li, Shangding Gu, Huichi Zhou, Shuqing Shi, Fei Ni, Shuo Lu, Weicheng Meng, Kang Li, Jin Wu, Kang Zhao, Shangmin Guo, Gen Li, Yongqiang Tang, Zhizhong Zhang, Yuan Xie, and Heng Qu. HarnessPAI: An Evolving Harness for Physical AI, 2026. URL https://arxiv.org/abs/2609.29166. arXiv preprint arXiv:2609.29166.

[221] Jiaheng Hu, Roberto Martin-Martin, Peter Stone, Rocky Duan, Zhenyu Jiang, and Guanya Shi. SimEX: Simulation-Integrated Robotics AutoResearch, 2026. URL https://arxiv.org/abs/2609.38982. arXiv preprint arXiv:2609.38982.

[222] Kaiyuan Chen, Shuangyu Xie, Letian Fu, Justin Yu, William Pacini, Sandeep Bajamahal, Hudson Kim, Jaimyn Drake, Daehwa Kim, Haoru Xue, Jonathan Francis, Christian Juette, Peter Schaldenbrand, Muhammet Yunus Seker, Ruwan Wickramarachchi, Uksang Yoo, Guanzhi Wang, Adithyavairavan Murali, Balakumar Sundaralingam, S. Shankar Sastry, Spencer Huang, Yuke Zhu, Linxi "Jim" Fan, and Ken Goldberg. GaP: A Graph-as-Policy Multi-Agent Self-Learning Harness For Variational Automation Tasks. *arXiv preprint arXiv:2607.05369*, 2026. URL https://arxiv.org/abs/2607.05369.

[223] Jorge Mendez-Mendez, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Embodied Lifelong Learning for Task and Motion Planning. In *Conference on Robot Learning*, pages 2134–2150, 2023. URL https://proceedings.mlr.press/v229/mendez-mendez23a.html.

[224] Yunhao Zhao, Zhenyang Ni, Haoyang Chen, Ruohan Zhang, and Qi Zhu. Skills in weights, memory in code: Hybrid learning for memory-dependent robot manipulation, 2026. URL https://arxiv.org/abs/2608.09410.

[225] Archit Sharma, Ahmed M. Ahmed, Rehaan Ahmad, and Chelsea Finn. Self-Improving Robots: End-to-End Autonomous Visuomotor Reinforcement Learning. In *Conference on Robot Learning*, pages 3292–3308, 2023. URL https://proceedings.mlr.press/v229/sharma23b.html.

[226] Jiazhi Yang, Kunyang Lin, Jinwei Li, Wencong Zhang, Tianwei Lin, Longyan Wu, Zhizhong Su, Hao Zhao, Ya-Qin Zhang, Li Chen, Ping Luo, Xiangyu Yue, and Hongyang Li. Rise: Self-improving robot policy with compositional world model. *arXiv preprint arXiv:2602.11075*, 2026. doi: 10.48550/arXiv.2602.11075. URL https://arxiv.org/abs/2602.11075.

[227] Varun Giridhar, Anant Khandelwal, Jeremy A. Collins, Ignat Georgiev, and Animesh Garg. Beyond imitation: Self-improving robot policies via off-policy q-planning, 2026. URL https://arxiv.org/abs/2608.21204.

[228] Gaojing Zhang, Changwen Chen, Zichen Luo, Jialiang Li, Yuhan Wang, Yifei Xie, and Wenzhao Lian. BioAgent: Self-Directed Skill Acquisition and Capability Growth for Long-Horizon Laboratory Protocols, 2026. URL https://arxiv.org/abs/2610.41383. arXiv preprint arXiv:2610.41383.

[229] Jiaru Bai, Yuchi Zhao, Han Hao, Elia Savino, Miroslav Bogdanovic, Derk H. W. ten Klooster, Simone Pilon, Timothy Noël, Yang Cao, Varinia Bernales, and Alán Aspuru-Guzik. Self-Building Labs, 2026. URL https://arxiv.org/abs/2610.23826. arXiv preprint arXiv:2610.23826.

[230] Huihan Liu, Soroush Nasiriany, Lance Zhang, Zhiyao Bao, and Yuke Zhu. Robot Learning on the Job: Human-in-the-Loop Autonomy and Learning During Deployment. *The International Journal of Robotics Research*, 44(10–11):1727–1742, 2025. doi: 10.1177/02783649241273901. URL https://doi.org/10.1177/02783649241273901.

[231] Ryan Hoque, Lawrence Yunliang Chen, Satvik Sharma, Karthik Dharmarajan, Brijen Thananjeyan, Pieter Abbeel, and Ken Goldberg. Fleet-DAgger: Interactive Robot Fleet Learning with Scalable Human Supervision. In *Conference on Robot Learning*, pages 368–380, 2023. URL https://proceedings.mlr.press/v205/hoque23a.html.

[232] Huihan Liu, Yu Zhang, Vaarij Betala, Evan Zhang, James Liu, Crystal Ding, and Yuke Zhu. Multi-Task Interactive Robot Fleet Learning with Visual World Models. In *Conference on Robot Learning*, pages 4286–4313, 2025. URL https://proceedings.mlr.press/v270/liu25g.html.

[233] Yi Wang, Xinchen Li, Pengwei Xie, Pu Yang, Buqing Nie, Yunuo Cai, Qinglin Zhang, Chendi Qu, Jeffrey Wu, Jianheng Song, Xinlin Ren, Jingshun Huang, Mingjie Pan, Siyuan Feng, Zhi Chen, and Jianlan Luo. Learning While Deploying: Fleet-Scale Reinforcement Learning for Generalist Robot Policies. *arXiv preprint arXiv:2605.00416*, 2026. URL https://arxiv.org/abs/2605.00416.

[234] Xue Qin, Simin Luan, John See, Zeyd Boukhers, Cong Yang, and Zhijun Li. Governed Capability Evolution: Lifecycle-Time Compatibility Checking and Rollback for AI-Component-Based Systems, with Embodied Agents as Case Study. *arXiv preprint arXiv:2604.08059v5*, 2026. URL https://arxiv.org/abs/2604.08059v5.

[235] Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, Yuan Xie, Heng Qu, Kun Shao, and Jian Luan. HarnessX: A Composable, Adaptive, and Evolvable Agent Harness Foundry. *arXiv preprint arXiv:2606.14249*, 2026. URL https://arxiv.org/abs/2606.14249.

[236] Xue Qin, Simin Luan, John See, Cong Yang, and Zhijun Li. EmbodiedGovBench: A Benchmark for Governance, Recovery, and Upgrade Safety in Embodied Agent Systems. *arXiv preprint arXiv:2604.11174*, 2026. URL https://arxiv.org/abs/2604.11174.

[237] Borui Kang, Jinrui Gu, Junhan Lv, Wenbin Li, Lei Wang, and Yang Gao. Harness continual learning: Continual adaptation beyond model parameters, 2026. URL https://arxiv.org/abs/2608.19013.

[238] Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, Dakuo Wang, Xiang Zhang, Benoit Dumoulin, Cihang Xie, Yuyin Zhou, Suhang Wang, and Hanqing Lu. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving llm agents. *arXiv preprint arXiv:2605.30621*, 2026. doi: 10.48550/arXiv.2605.30621. URL https://arxiv.org/abs/2605.30621.

[239] Xujun Li, Kehan Zheng, Mingyuan Zhao, Yize Geng, Jinfeng Zhou, Qi Zhu, Fei Mi, Lifeng Shang, Minlie Huang, and Hongning Wang. You live more than once: Towards hierarchical skill meta-evolving. *arXiv preprint arXiv:2605.28390*, 2026. doi: 10.48550/arXiv.2605.28390. URL https://arxiv.org/abs/2605.28390.

[240] Shuai Shao, Kangning Zhang, Qingyao Li, Shijian Wang, Hao Wang, Wenxiang Jiao, Yuan Lu, Yi Guo, Weiwen Liu, and Weinan Zhang. Harness-r1: Learning to edit executable runtime harnesses from agent failure trajectories. *arXiv preprint arXiv:2608.02276*, 2026. doi: 10.48550/arXiv.2608.02276. URL https://arxiv.org/abs/2608.02276.

[241] Jingdong Zhang, Jionghao Wang, Zhengming Yu, Yancheng Cai, Wei Chow, Chris Choy, Zhiyang Dou, Hang Hua, Jiachen Li, Zhixuan Liu, Chen Wang, Songlin Wei, Siyuan Yang, Yu Yuan, Xiaohang Zhan, Yucheng Zhang, Shuhong Zheng, Xin Li, Lingjie Liu, Wojciech Matusik, Zhengzhong Tu, and Wenping Wang. Flex-RSI: Benchmarking Recursive Self-Improvement of Frontier Agents, 2026. URL https://arxiv.org/abs/2611.08946. arXiv preprint arXiv:2611.08946.

[242] Lingjun Mao, Jinglin Cao, Lukun He, Wenpeng Xu, Yuchen Yan, Hengyu Jin, Zijun Zhang, Jinzhou Tang, Songyao Jin, Ziche Liu, Yifei Shao, Junbo Huang, Fang Nan, Ruobing Han, Ziqiao Xi, Ziming Xu, Shuang Liang, Sibo Zhu, Wenyi Wu, Xinyue Wang, Yiming Huang, Xinkai Zou, Gao Zhu, Kun Zhou, and Biwei Huang. RobotAgentBench: Towards Open-world Robot Autonomy, 2026. URL https://arxiv.org/abs/2610.34649. arXiv preprint arXiv:2610.34649.

[243] 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*, volume 36, 2023. doi: 10.52202/075280-1939. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/8c3c666820ea055a77726d66fc7d447f-Abstract-Datasets_and_Benchmarks.html.

[244] Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-Scale Simulation of Everyday Tasks for Generalist Robots. *arXiv preprint arXiv:2406.02523*, 2024. URL https://arxiv.org/abs/2406.02523.

[245] Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A Large-Scale Simulation Framework for Training and Benchmarking Generalist Robots. *arXiv preprint arXiv:2603.04356*, 2026. URL https://arxiv.org/abs/2603.04356.

[246] Yao Mu, Tianxing Chen, Zanxin Chen, Shijia Peng, Zhiqian Lan, Zeyu Gao, Zhixuan Liang, Qiaojun Yu, Yude Zou, Mingkun Xu, Lunkai Lin, Zhiqiang Xie, Mingyu Ding, and Ping Luo. RoboTwin: Dual-Arm Robot Benchmark with Generative Digital Twins. *arXiv preprint arXiv:2504.13059*, 2025. URL https://arxiv.org/abs/2504.13059.

[247] Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan-Ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. RoboTwin 2.0: A Scalable Data Generator and Benchmark with Strong Domain Randomization for Robust Bimanual Robotic Manipulation. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 13673–13699. PMLR, 2026. URL https://proceedings.mlr.press/v306/chen26j.html.

[248] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Wensi Ai, Benjamin Martinez, Hang Yin, Michael Lingelbach, Minjune Hwang, Ayano Hiranaka, Sujay Garlanka, Arman Aydin, Sharon Lee, Jiankai Sun, Mona Anvari, Manasi Sharma, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R. Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Yunzhu Li, Silvio Savarese, Hyowon Gweon, C. Karen Liu, Jiajun Wu, and Li Fei-Fei. BEHAVIOR-1K: A Human-Centered, Embodied AI Benchmark with 1,000 Everyday Activities and Realistic Simulation. *arXiv preprint arXiv:2403.09227*, 2024. URL https://arxiv.org/abs/2403.09227.

[249] Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. EmbodiedBench: Comprehensive Benchmarking Multi-modal Large Language Models for Vision-Driven Embodied Agents. *arXiv preprint arXiv:2502.09560*, 2025. URL https://arxiv.org/abs/2502.09560.

[250] Yuhao Chen, Zhihao Zhan, Xiaoxin Lin, Zijian Song, Hao Liu, Qinhan Lyu, Yubo Zu, Xiao Chen, Zhiyuan Liu, Tao Pu, Tianshui Chen, Keze Wang, Liang Lin, and Guangrun Wang. RADAR: Benchmarking Vision-Language-Action Generalization via Real-World Dynamics, Spatial-Physical Intelligence, and Autonomous Evaluation. *arXiv preprint arXiv:2602.10980*, 2026. URL https://arxiv.org/abs/2602.10980.

[251] Alex S. Huang, Jiahui Zhang, Shiqing Tang, and Yu Xiang. VLA-REPLICA: A Low-Cost, Reproducible Benchmark for Real-World Evaluation of Vision-Language-Action Models. *arXiv preprint arXiv:2605.20774*, 2026. URL https://arxiv.org/abs/2605.20774.

[252] Suhwan Choi, Yunsung Lee, Yubeen Park, Chris Dongjoo Kim, Ranjay Krishna, Dieter Fox, and Youngjae Yu. vla-eval: A Unified Evaluation Harness for Vision-Language-Action Models. *arXiv preprint arXiv:2603.13966*, 2026. URL https://arxiv.org/abs/2603.13966.

[253] Yiheng Lyu, Xueying Jiang, Wenhao Li, Shijian Lu, and Gongjie Zhang. CodeActionBench: Evaluating Agentic Code-as-Policy for Embodied Manipulation, 2026. URL https://arxiv.org/abs/2609.33807. arXiv preprint arXiv:2609.33807.

[254] Zhiqin Yang, Chenxin Li, Xiaomeng Hu, Yibin Liu, Weidong Huang, Jiankai Sun, Haitao Li, Zijian Wu, Yuzhi Huang, Fanding Huang, Hanwen Sun, Jiashun Liu, Jingqi Tong, Mingxin Huang, Shaoli Hu, Shijue Huang, Tianyi Bai, Xinyuan Wang, Yunlong Lin, Zhengyang Tang, Zhexin Zhang, Zhuo Chen, Xierui Song, Juntao Dai, Boyuan Chen, Jiaming Ji, Fangneng Zhan, Mengkang Hu, Wei Xue, Yonggang Zhang, Han Hu, Tsung-Yi Ho, and Yike Guo. RobotWorld: Benchmarking Multimodal Agents for Robot Use Across Diverse Tasks and Embodiments, 2026. URL https://arxiv.org/abs/2610.10409. arXiv preprint arXiv:2610.10409.

[255] Liu Renhang, Navonil Majumder, Tej Deep Pala, and Soujanya Poria. RoboQuest: Generalist Physical Agents that Search, Inspect and Test, 2026. URL https://arxiv.org/abs/2610.10388. arXiv preprint arXiv:2610.10388.

[256] Feng Chen, Pei Zhou, Lixuan He, Yunchao Yao, Li Sun, Shenghua Gao, Mingyu Ding, Kaifeng Zhang, Kaichen Zhou, and Yi Ma. Embodied Playground: A Long-Horizon Agentic Robotics Benchmark for Physical, Strategic, and Spatial Reasoning, 2026. URL https://arxiv.org/abs/2610.27039. arXiv preprint arXiv:2610.27039.

[257] Zhi Han, Zihan Guo, Chenxi Zeng, Zicai Cui, Yue Xin, Ming Zhou, Dongzhan Zhou, Weinan Zhang, and Yang Li. Embodied Skill Bench: Benchmarking How Well Agents with Embodied Skills Perform on Physical Tasks, 2026. URL https://arxiv.org/abs/2610.16238. arXiv preprint arXiv:2610.16238.

[258] Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Haoran Lu, Weijie Wan, Baijun Chen, Songling Liu, Haowen Yan, Honghao Su, Zhiyang Dou, Kaixuan Wang, Dandan Zhang, Yunze Liu, Yan Qin, Qiwei Liang, Qiwei Wu, Zijian Lin, Wenwei Lin, Yuran Wang, Minghua He, Tianshu Wu, Ruihai Wu, Jingquan Zhou, Kai-Chong Lei, Haibao Yu, Yuanfeng Ji, Weiyang Jin, Guanyu Lin, Xiaofan Li, Qi Xiong, Renjing Xu, Zhongyu Li, Wenhao Chai, Enze Xie, Ziwei Wang, Yao Mu, Hao Dong, Wojciech Matusik, Mingyu Ding, Wenbo Ding, Ping Luo, and Masayoshi Tomizuka. RoboDojo: A Unified Sim-and-Real Benchmark for Comprehensive Evaluation of Generalist Robot Manipulation Policies, 2026. URL https://arxiv.org/abs/2607.04434. arXiv preprint arXiv:2607.04434.

[259] Tan Sang Nguyen, Yi Ru Wang, Shailesh Shailesh, Gang Yang, Jiamo Hu, Jianuo Jiang, Haoquan Fang, Zhongzheng Ren, Zhongyi Luo, Ian Chong, Aaron Tung, Santiago Pravisani, Minghui Lin, Yao Mo, Ishneet Singh, Benjamin Tee, Harold Soh, Siddhartha Srinivasa, Dieter Fox, David Hsu, Haoming Song, Maxime Ellerbach, Thomas Wolf, Pepijn Kooijmans, Ruijie He, Michael Cho, and Jiafei Duan. RoboColosseum: Evaluating Physical Intelligence at Scale In-The-Wild, 2026. URL https://arxiv.org/abs/2611.02497. arXiv preprint arXiv:2611.02497.

[260] Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Mottaghi, Priyam Parashar, Siddharth Patki, Ishita Prasad, Xavier Puig, Akshara Rai, Ram Ramrakhya, Daniel Tran, Joanne Truong, John M. Turner, Eric Undersander, and Tsung-Yen Yang. PARTNR: A Benchmark for Planning and Reasoning in Embodied Multi-agent Tasks. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a3cf318fbeec1126da21e9185ae9908c-Abstract-Conference.html.

[261] Yang Li, Chen Zhao, Zhuoran Wang, Jiankang Wang, Chao Shao, Yihan Lin, Haitao Shen, and Jing Zhang. RoboRecover: Benchmarking Robot Policy Recovery under Execution Deviations. *arXiv preprint arXiv:2609.28952*, 2026. URL https://arxiv.org/abs/2609.28952.

[262] Nana Yaw Owusu Ofori-Ampofo, Samira Ebrahimi Kahou, and Joseph Thekinen. REBOOT: From Failure to Recovery - A Dataset and Benchmark for Precision Assembly. *arXiv preprint arXiv:2609.22591*, 2026. URL https://arxiv.org/abs/2609.22591.

[263] Sheng Yin, Xianghe Pang, Yuanzhuo Ding, Menglan Chen, Yutong Bi, Yichen Xiong, Wenhao Huang, Zhen Xiang, Jing Shao, and Siheng Chen. SafeAgentBench: A Benchmark for Safe Task Planning of Embodied LLM Agents. *arXiv preprint arXiv:2412.13178*, 2024. URL https://arxiv.org/abs/2412.13178.

[264] Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624:570–578, 2023. doi: 10.1038/s41586-023-06792-0. URL https://doi.org/10.1038/s41586-023-06792-0.

[265] Zachary Ravichandran, Fernando Cladera, Jason Hughes, Varun Murali, M. Ani Hsieh, George J. Pappas, Camillo J. Taylor, and Vijay Kumar. Deploying Foundation Model-Enabled Air and Ground Robots in the Field: Challenges and Opportunities. *arXiv preprint arXiv:2505.09477*, 2025. URL https://arxiv.org/abs/2505.09477.