Embodied Agentic Harnesses for Physical Intelligence A Survey

Yu Huang*, Yue Chen&, Zijiang Yang#, Gary Ding^

*: Institute of Advanced Technology, University of Science and Technology of China, China

#: School of Computer Science & Technology, University of Science and Technology of China, China

&: Futurewei Technology Inc., San Jose, USA

^: Dartmouth College

Abstract

Embodied agents depend on more than the capabilities of a language, vision-language, or action model. They require a runtime that grounds observations, coordinates executable capabilities, monitors outcomes, and manages intervention when physical execution diverges from a plan. This survey examines embodied agentic harnesses as this runtime control plane. We distinguish four logical responsibilities—reasoning, runtime governance, skill execution, and hardware interfacing—and separate essential execution services from optional prediction, persistent memory, and adaptation mechanisms. A contract-based formulation connects skill preconditions, resource ownership, interruption, postconditions, and trace collection without assuming that every system implements an identical stack. We organize the literature by mechanisms rather than publication period: grounding and composition, state and memory, scheduling and coordination, verification and recovery, and adaptation. Complementary descriptors record the execution interface, intervention timescale, adaptation target, and evidence setting. The synthesis connects LLM and agent foundations, VLA policies, world models, and predictive-action models to affordance-based planning, programmatic control, memory-guided orchestration, agent operating systems, and evolving runtime critics. It distinguishes operational harnesses from infrastructure for robot learning and deployment. We further distinguish semantic task verification, invocation authority, and physical safety assurance, which require different evidence. The resulting synthesis identifies execution contracts, compositional evaluation, bounded adaptation, and explicit human authority as priorities for reliable long-horizon physical intelligence.

Keywords: Embodied intelligence; Agentic harness; Runtime architecture; Skill orchestration; Memory; Execution contracts; Runtime assurance; Cross-embodiment control

Chapter 1 Introduction and Scope

1.1 From model capability to system reliability

Language and vision-language models support instruction interpretation and task reasoning, while learned action policies provide embodied execution capabilities [1–5]. These contributions address different parts of a robotic system. A VLA may generate actions from multimodal inputs; a world model predicts consequences or supports planning. Neither category alone specifies how a deployed agent should arbitrate competing goals, invalidate stale observations, determine whether an action succeeded, or transfer control after a failure [6–8].

Foundation-model research explains why broad pretraining can supply transferable representations while also propagating shared weaknesses into downstream applications [9, 10]. Multimodal foundation models extend this capability to visual interpretation and tool-mediated assistance [11, 12]. Agent surveys examine how perception, planning, memory, and action are assembled around these models [13, 14]; embodied-AI surveys add the constraints of robot morphology, physical interaction, and sim-to-real transfer [15, 16]. An embodied harness occupies the intersection: it turns model outputs into accountable execution and returns grounded evidence to subsequent decisions.

This distinction becomes consequential over long horizons. A plausible instruction can refer to an unreachable object; a locally successful grasp can leave the next policy outside its operating distribution; and a completed

software call can correspond to an unfinished physical task. Affordance grounding in SayCan, executable programs in Code as Policies, and assertions in ProgPrompt address complementary parts of this gap [2, 17, 18]. Memory-guided orchestration and explicit execution sessions address the same gap at system level [19–21]. The common problem is maintaining correspondence between intentions, observable state, and executable capabilities throughout interaction.

Physical action also changes the meaning of recovery. Rolling back software state cannot restore a dropped object or undo a collision. Conversely, not every foundation-model robot is open loop: feedback can exist within the policy, the planner, or an external runtime. The analytical question is where feedback enters, what it can change, and which failures remain outside its authority. This survey therefore examines runtime mechanisms and their interfaces instead of treating the presence of a large model as evidence of an unsafe or open-loop architecture.

1.2 Definition and boundaries

We define an embodied agentic harness as the runtime control plane that mediates between an agent's proposals and embodied execution by maintaining execution-relevant state, governing capability invocation, and returning monitored outcomes and intervention signals. Its minimal functional boundary includes an execution interface, lifecycle management, and a feedback path. These functions can be implemented in one process or distributed across services; they need not form a separately named software package.

Persistent memory, learned prediction, code generation, and online adaptation are optional mechanisms within this definition. Requiring all of them would exclude useful harnesses and conflate a reference architecture with a universal implementation. Likewise, the harness can request a stop or install a controller-level constraint, but physical stopping and low-level stabilization belong to the mechanisms actually connected to the robot. An LLM judgment is not itself a certified safety interlock [22–24].

We include both integrated runtimes and methods that supply a clearly identifiable runtime mechanism. Behavior trees and task-and-motion planning are architectural foundations rather than complete foundation-model harnesses [25–28]. ACT and Diffusion Policy are execution policies, while ROS 2 supplies middleware infrastructure [4, 5, 23]. Roadmaps and critical reviews inform conceptual boundaries but are not experimental systems [8, 29]. Table 1 fixes these distinctions before the method comparison.

This is a focused narrative review of mechanisms and their interfaces, with robotic manipulation as the main empirical setting and navigation, multi-robot systems, and general language agents used where they clarify transferable principles. We distinguish research systems from surveys and conceptual proposals, and cite preprints as preprints. The comparisons establish reported design choices and evidence boundaries; they do not constitute a pooled meta-analysis or a complete census of all embodied-AI methods.

Table 1. Boundaries of the surveyed components

Component	Primary responsibility	Boundary and examples
Reasoning model	Propose goals, plans, code, or action units	Proposal quality does not establish execution correctness [1–3, 17].
Harness runtime	Govern invocation and execution feedback	State, contracts, lifecycle, coordination, monitoring [21, 30, 31].
Skill or policy	Implement executable behavior	ACT and Diffusion Policy are action components, not complete harnesses [4, 5].
World model	Predict consequences or support planning	Optional service; predictions require uncertainty handling [6–8].

Component	Primary responsibility	Boundary and examples
Middleware and hardware	Communicate, control, and actuate	ROS 2 provides infrastructure; timing assurance depends on deployment [23].
Assurance mechanism	Constrain or verify a specified property	Physical constraints, permissions, and semantic outcomes require different checks [22, 32, 33].

1.3 Failure modes and survey contributions

Three recurring failures motivate the organization of this review. Grounding failures arise when semantic references, geometric state, and executable parameters disagree. Coordination failures arise when state becomes stale, asynchronous tasks contend for the same actuator, or a policy handoff violates the receiving policy's preconditions. Assurance failures arise when a process return, a model's confidence, or an authorization decision is treated as proof of physical success or safety. Persistent memory can reduce repeated mistakes, but stale or unsupported memory can also reproduce them [20, 30–33].

The survey contributes a responsibility-based architecture, an execution-contract formulation, a mechanism-based taxonomy, and an evaluation framework. Chapter 2 defines interfaces and assumptions; Chapter 3 relates physical constraints to intervention mechanisms; Chapter 4 compares methods within a unified classification; Chapter 5 discusses implementation choices; Chapter 6 examines evidence and unresolved challenges; and Chapter 7 synthesizes the research agenda. The taxonomy is an analytical framework, not a claim that its categories are mutually exclusive or that the surveyed systems implement all proposed services.

1.4 Relationship to neighboring surveys

The surrounding literature can be read along three complementary axes: learned capability, agent organization, and execution responsibility. LLM surveys primarily describe pretraining, adaptation, utilization, and evaluation [10]. Reviews of chain-of-thought and agentic reasoning explain how intermediate reasoning, search, tool use, and feedback extend a single model invocation [34, 35]. These mechanisms can improve the proposal process, but a reasoning trace is neither a robot-state estimate nor an execution record. The present survey follows the proposal through admission, physical action, verification, and recovery.

Agent taxonomies provide useful but differently scoped organizing principles. Li groups agent workflows around tool use, planning, and feedback learning, while Luo et al. connect agent construction, collaboration, and evolution [36, 37]. Proposed levels of AI agents organize broad capability progression [38]; the critique by Xing et al. instead questions whether externally assembled competence constitutes internalized agency [39]. We treat these as analytical positions rather than settled maturity standards. Our use of agentic describes an interactive execution system and does not assert consciousness, endogenous goals, or a universal hierarchy of autonomy.

VLA surveys offer several views of the policy side of this system. Ma et al. cover model components, control policies, and higher-level planning [40]; Zhong et al. organize interfaces by action-token representation [41]; Shao et al. distinguish monolithic and hierarchical architectures [42]; and Zhang et al. compare autoregressive, diffusion, reinforcement-based, and hybrid approaches [43]. Kawaharazuka et al. connect models to hardware, data collection, and deployment [44]. These views motivate our execution-interface descriptor, but they do not determine the runtime mechanism: the same policy family can be deployed with different memory, scheduling, and recovery designs.

World-model surveys likewise classify different objects. Zhu et al. connect video generation, driving, and autonomous agents, whereas Ding et al. distinguish representations of the current world from prediction of future dynamics [45, 46]. Li et al. analyze functionality, temporal modeling, and spatial representation, and Hou et al. emphasize coupling to robot policies, learned simulation, and evaluation [47, 48]. Sun et al. examine modular, sequential, and unified integration with action and navigation; Tan et al. distinguish planning, action generation,

data synthesis, and simulation roles [49, 50]. A prediction service must therefore be identified by its role in the deployed loop, rather than inferred from the phrase world model.

Harness surveys make the external execution system an explicit research object. He et al. propose control, agency, and runtime as a decomposition; Meng et al. describe execution, tool, context, state, lifecycle, and evaluation components; and Li et al. make observability and governance explicit in a seven-layer taxonomy [51–53]. Externalization and code-centered surveys explain how memory, skills, protocols, and executable artifacts structure this infrastructure [54, 55]. Guo et al. connect harness design to task completion, while Dong et al. link external harnesses with model optimization across long horizons [56, 57]. Our extension concerns the physical boundary: action irreversibility, observation age, embodiment-specific feasibility, and recovery that must itself be executed and checked.

Table 2 locates this review relative to these neighboring surveys. It is a comparison of scope, not a ranking of survey quality. Model learning, system construction, and deployment governance remain coupled research problems; distinguishing their evidence makes that coupling easier to analyze.

Table 2. Scope of neighboring surveys and the embodied harness perspective

Literature perspective	Organizing question	Contribution to this review
LLM and agent surveys [10, 13, 14, 36]	How are models, tools, planning, and feedback assembled?	Background for the reasoning and interaction loop.
VLA architecture and action interfaces [40–42, 44]	How is multimodal input converted into executable behavior?	Policy interface and embodiment assumptions.
World models and robot learning [46–48]	What is predicted and how is prediction used?	Predictive-service role and evidence limitations.
Predictive action and WAM surveys [49, 50, 58, 59]	How are future representations coupled to action?	Training versus deployment; latent versus rendered futures.
General agent harnesses [51–53, 56]	How is persistent execution controlled and observed?	Runtime services and governance boundaries.
Memory, skills, and long horizons [57, 60–62]	What persists and how is it reused or revised?	Freshness, provenance, commitments, and lifecycle.
This embodied harness review	How do composed capabilities remain accountable during physical interaction?	Contracts, intervention timing, recovery, and causal evaluation.

1.5 Model families and their runtime implications

The distinction between a policy and a harness should not imply that policies are necessarily memoryless or incapable of temporal reasoning. The π0 family illustrates how flow-based action generation, heterogeneous co-training, semantic subtask prediction, and richer context conditioning can change the scope of an executable model [63–65]. Reviews of VLA post-training and reinforcement learning show further routes for adapting perception, task understanding, and embodied behavior [66, 67]. The appropriate boundary is operational: which decisions are learned inside the model, which are managed by the surrounding system, and how failures cross that boundary.

World action models (WAMs) make predictive information useful for action generation, but definitions vary across surveys. Wang et al. emphasize joint future-state and action modeling and distinguish cascaded from joint architectures [58]. Shen et al. also include latent futures and video-generation-free predictive action reasoning [59]. We therefore use WAM as a predictive-action category and explicitly record the predicted variable, action

coupling, and deployment regime. It is not a successor stage that automatically replaces VLAs, nor does a video backbone by itself establish action-conditioned causal validity.

This distinction matters in concrete systems. Motus unifies multiple understanding, prediction, and action modes; mimic-video uses video-derived latent representations with an action decoder; and DreamZero jointly models future video and actions [68–70]. Cosmos Policy includes actions, future observations, and values within a video-model formulation, while LingBot-VA couples frame prediction and policy execution with observed feedback [71, 72]. These are different ways to construct a capability that the harness may invoke. Their internal predictive computation should not be counted again as an independent external verification mechanism.

Other designs weaken the link between predictive training and explicit imagination at deployment. VLA-JEPA uses future latent targets during pretraining; Being-H0.7 aligns a deployable prior with a training-only future-informed branch; and Fast-WAM retains video co-training while omitting future generation at inference [73–75]. BagelVLA instead interleaves linguistic reasoning, visual forecasting, and action generation [76]. DreamDojo provides an action-conditioned robot world model that can support simulation, evaluation, or planning [77]. Thus, the architecture must record whether prediction is an offline learning signal, a callable planning service, or part of the executing policy.

Figure 1 summarizes these relationships. Learned representations and policies supply capabilities; the harness decides how capabilities are invoked and how evidence changes the next decision. The division is logical: implementations may share networks, processes, or middleware, provided the corresponding responsibilities and assumptions remain identifiable.

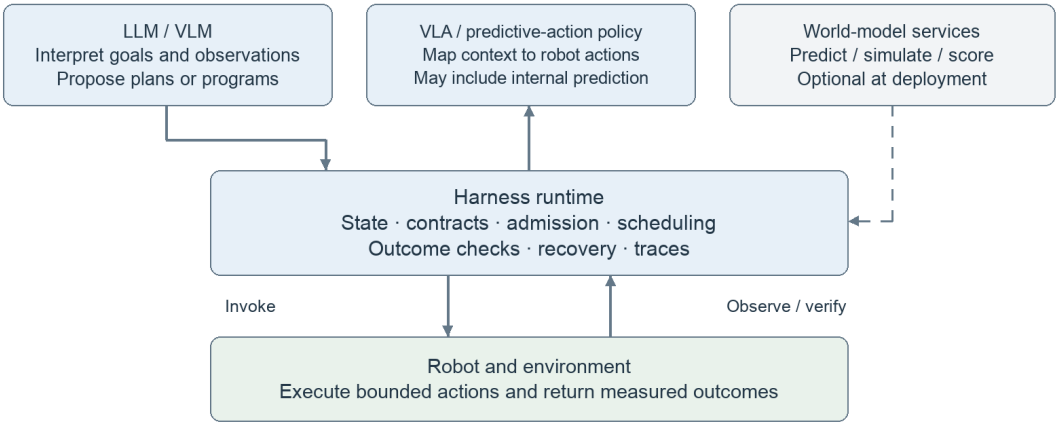

Figure 1. Model capabilities and the embodied execution loop. Dashed links denote optional prediction or experience services; the lower loop returns measured outcomes. A WAM may implement the policy, the predictive service, or both. The diagram is an author synthesis of model, agent, and runtime distinctions [13, 40, 47, 58, 59, 78].

Chapter 2 Architecture and Execution Contracts

2.1 Four logical responsibilities

The reference architecture separates reasoning, runtime governance, skill execution, and hardware interfacing, as illustrated in Figure 2. These are logical responsibilities, not four independent physical processes or fixed frequency bands. A VLM may choose fine-grained semantic actions, a VLA may operate as a callable primitive, and a conventional controller may execute the resulting command. The location of model inference therefore does not uniquely determine a system's architectural category [19, 79, 80].

The reasoning responsibility interprets goals and proposes plans, programs, or action units. The harness responsibility manages context, invocation contracts, task state, scheduling, outcome monitoring, and escalation. The execution responsibility implements bounded capabilities through learned policies, motion planners, or

feedback controllers. The hardware and middleware responsibility transports commands and measurements and implements device-facing control. Timing guarantees depend on the complete deployment, including scheduling, communication, drivers, and actuators; selecting ROS 2 alone does not establish hard real-time behavior [23].

The placement of the harness is itself debated. Lee et al. argue that robot middleware is positioned to coordinate control, computation, and communication, proposing output projection, execution isolation, and transfer to a baseline [78]. This is compatible with a logical runtime responsibility distributed through middleware, rather than requiring an additional monolithic process above ROS 2. Their proposed deployment profile is an architectural direction, not evidence that every middleware installation already enforces model-output or timing constraints.

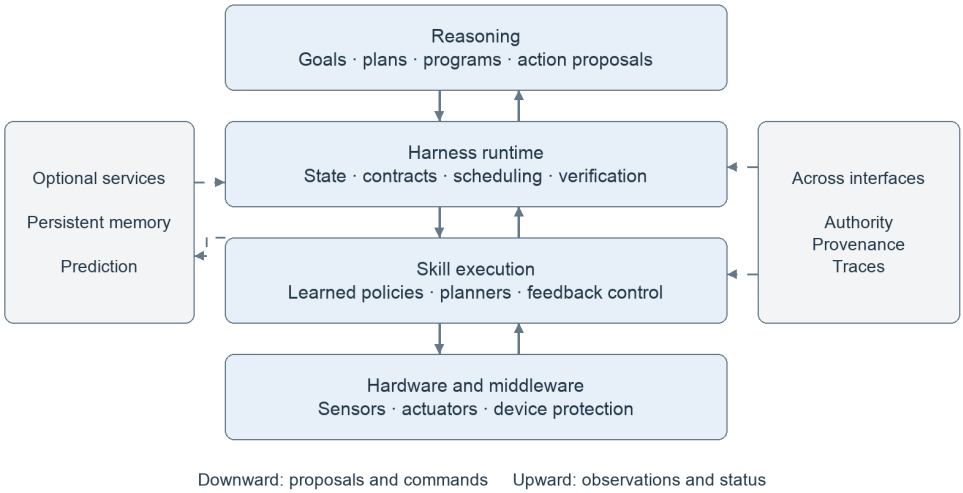

Figure 2. Four logical responsibilities and their information flows. Optional memory and prediction support the runtime; authority, provenance, and traces span interfaces. Logical layers do not specify deployment processes or frequencies. Author synthesis [17, 21–23, 30, 31].

The architecture also contains cross-cutting concerns. Authorization constrains which actor may invoke a capability; provenance links state estimates to observations; hardware health can narrow the feasible action envelope; and trace collection supports debugging and evaluation. These concerns should be expressed at the interfaces they affect. Assigning all of them to a single central block would obscure which component can enforce a decision and what happens when that component becomes unavailable.

2.2 A functional decomposition of the harness

For comparison, we use the descriptive tuple H = (O, L, M, P, V, K, G, E, D). O denotes observation and state handling; L lifecycle and scheduling; M persistent memory; P optional predictive services; V outcome and constraint evaluation; K the skill registry and invocation contracts; G intervention and authorization rules; E embodiment interfaces; and D diagnostics and execution traces. This decomposition organizes responsibilities. It is not a complete mathematical state space, an implementation checklist, or a proof of reliability.

The tuple extends prior harness decompositions by making embodied state, capability contracts, and intervention responsibilities explicit [51–53]. It should not be read as nine independent modules. For example, AEROS organizes capabilities into installable modules under a persistent agent and a policy-separated runtime, while Harnessing Embodied Agents emphasizes admission, monitoring, recovery, and human override [81, 82]. Both fit several entries of the tuple; comparison should examine the implemented interfaces and the reported simulation setting rather than reward a system for matching the number of components.

The distinction between K and the underlying policy library is essential. The harness records what a capability accepts, requires, returns, and allows during interruption; the skill implementation generates the physical behavior. Similarly, E specifies how a task-level or semantic command is grounded for a particular robot,

while the resulting trajectory remains subject to robot-specific feasibility and feedback control. World models support P when their predictions are useful, but predictive uncertainty prevents P from subsuming V or G [6–8, 22].

O and M have different lifetimes. A timestamped estimate used to execute the current action belongs to O; reusable spatial knowledge, interaction history, and capability experience belong to M. Scene-graph perception provides a structured basis for environmental state [83, 84]. ABot-AgentOS extends the persistent side with source-grounded multimodal graph memory [31], while PhyAgentOS exposes session state through structured files [21]. The storage format does not establish correctness: provenance, freshness, and conflict handling determine whether the state is actionable.

Persistent state includes more than retrieved memories. The always-on-agent literature also identifies task ledgers, permissions, commitments, provenance, and mutation histories as behaviorally consequential state [62]. For a robot, an interrupted placement task may need both a remembered goal and a fresh estimate of whether the gripper still holds the object. The first survives a session boundary; the second must be measured again. Treating both as an undifferentiated context buffer hides this distinction.

2.3 Skill contracts and lifecycle semantics

We represent a skill invocation contract as C = (I, P, R, B, Q, X, T). I specifies typed inputs; P preconditions; R resources and authority; B execution bounds; Q observable postconditions; X interruption and recovery behavior; and T the required trace. This is a proposed common description for comparing implementations, not an assertion that all surveyed systems expose this exact schema. A capability may bind a learned primitive, generated program, motion planner, or deterministic semantic-action interpreter [17, 30, 79, 80].

The word skill also spans different abstractions. In general language-agent systems it may denote a package of instructions, code, and resources loaded on demand [61]. In robotics it may denote a policy or controller with physical preconditions and termination behavior. A portable textual package therefore does not imply portable motor behavior. The invocation contract connects these uses by specifying the callable artifact, required observations, execution authority, and observable result, while leaving embodiment-specific realization to the implementation.

An illustrative lifecycle is PROPOSED → ADMITTED → RUNNING → VERIFYING → SUCCEEDED, with explicit transitions to REJECTED, FAILED, CANCELLED, or ESCALATED. Admission requires current evidence for preconditions and resources. Completion of the underlying process triggers verification rather than automatic success. An interrupted action must relinquish resources only after reaching the contract's interruption condition, and a resumed action must revalidate its preconditions. Physical interruption can require holding an object or decelerating before ownership is transferred.

ProgPrompt embeds assertions into situated task programs [18]; Thea makes outcome evaluation available through exit-code semantics [32]; EmbodiedSkills explicitly separates proposal, prerequisite checking, bounded execution, and outcome verification [79]. These mechanisms can be compared through the same lifecycle despite using different program representations. Their shared contribution is making execution status observable and actionable rather than treating an issued command as a completed subgoal.

2.4 Three distinct verification questions

The first question is whether a proposed action is relevant and executable in the estimated state. Affordance scores and task-and-motion constraints inform this question [2, 27, 28]. The second is whether the caller is authorized to invoke it: typed authority and evidence-dependent guards address permission rather than dynamics [33]. The third is whether physical execution remains within a specified operating envelope. A controller-level safety filter can address this question under explicit model, sensing, and feasibility assumptions [22]. These checks may be composed, but their conclusions are not interchangeable.

For a state estimate $\hat{x}$ and candidate command u, an illustrative admission predicate is $A(\hat{x},u) = P(\hat{x},u) \wedge Auth(u) \wedge Fresh(\hat{x}) \wedge Resource(u)$. Physical constraints can be expressed separately as $c_j(\hat{x},u) \leq 0$ for each monitored bound. For example, a torque bound is $|\tau_i| - \tau_{i,max} \leq 0$. Under partial observation, evaluating these predicates at an estimate is evidence for admission, not proof that all possible physical states are safe.

Runtime intervention additionally requires a model of what can still be prevented. Detection latency, communication delay, braking behavior, and available fallback actions jointly determine whether a response is timely. Outcome verification has a different role: it asks whether the intended postcondition was achieved. LLM-based plan reviewers can reject or escalate proposals [24], but an accepted plan does not certify collision avoidance. Figure 3 makes these distinct decision points explicit.

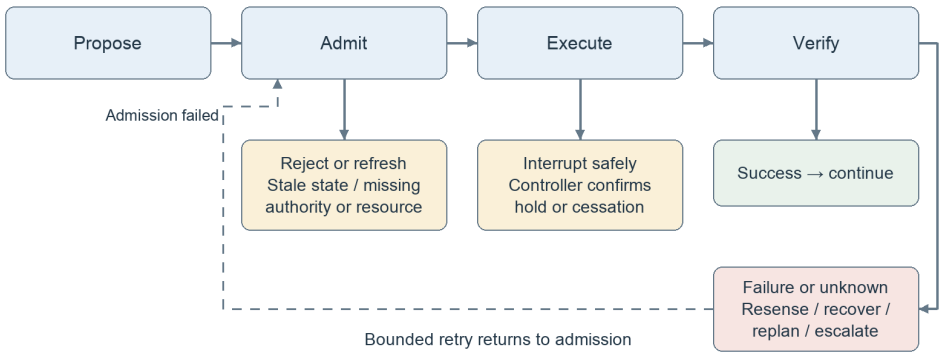

Figure 3. Execution lifecycle with rejection, interruption, and outcome branches. A failed or unknown outcome returns to admission only within the allowed recovery budget. Protection is enforced by the relevant controller or hardware path. Author synthesis [18, 22, 25, 32, 33, 79].

Chapter 3 Physical Constraints and Runtime Responses

3.1 Irreversibility and bounded intervention

A physical action may be irreversible even when its software invocation is cancellable. The harness therefore needs admission checks before commitment, monitoring during execution, and outcome checks afterward. A predictive rollout can help compare candidate actions, but model error can invalidate its forecast. Independent actuator limits, protected operating regions, and an application-specific stopping strategy remain necessary where the deployment requires them [22, 85, 86].

Object-centric manipulation reviews distinguish affordance perception, policy learning, and task-oriented reasoning, while broader manipulation taxonomies separate planning representations from learning-based control [87, 88]. This explains why semantic correctness is insufficient at contact: an instruction can identify the correct object yet omit friction, grasp stability, or reachable approach geometry. Runtime checks should be attached to the physical dependency actually needed by the next action, with uncertainty carried forward when that dependency cannot be observed.

Recovery should be selected according to the observed failure and remaining feasible actions. A failed perception check may justify resensing; a failed grasp may justify restaging; excessive contact force may require a controller-mediated release or stop. An unconditional retract command can itself be hazardous in constrained contact. We therefore treat recovery as another contracted capability, with its own preconditions, budget, and outcome criteria. Success rates and recovery delays must be measured for a stated task and hardware configuration, rather than assigned universally to the harness abstraction.

3.2 Asynchrony and shared physical resources

Reasoning, perception, and motor control operate at different and often variable timescales. Observation buffers should retain timestamps, coordinate frames, and validity intervals; proposals should record the state version on which they depend. At dispatch, the runtime can accept the proposal, refresh its grounding, or reject it because the scene has changed. Merely increasing planner throughput does not solve stale-state execution.

Spatial understanding provides another source of mismatch. Work on 4D reconstruction distinguishes static geometry, dynamic scenes, interaction, and physical constraints [89]; spatial-reasoning reviews separate perceptual recognition from geometric reasoning and embodied use [90, 91]. The SIBench analysis reports gaps between basic perception and harder spatial understanding and planning [92]. For a harness, the implication is to preserve coordinate frames, object identities, and temporal provenance rather than treating a fluent visual description as an interchangeable substitute for actionable state.

TypeGo combines asynchronous planning with typed subsystem arbitration, preemption, and speculative skill streaming [30]. Behavior-tree execution supplies a different form of reactive orchestration through repeated condition evaluation [25, 26]. The common comparison is how each approach handles event arrival, cancellation, and resource ownership. Speculation reduces waiting only when queued actions remain valid; reactive ticks improve responsiveness only when the required observations and interruptible actions are available.

A useful latency budget separates observation age, reasoning time, queueing, dispatch, and actuator response. Mean task latency and worst-case interruption delay answer different questions. Designs should state what happens when a budget expires: continue a bounded local policy, enter a hold condition, stop through the controller, or escalate. No universal frequency threshold separates a harness from a controller because both responsibilities may contain multiple loops.

Figure 4 shows why admission must use current state rather than the state available when reasoning began. The critical event is a violated dependency, such as a moved object or a changed resource owner, not the elapsed time alone. A fresh but irrelevant observation does not resolve that mismatch.

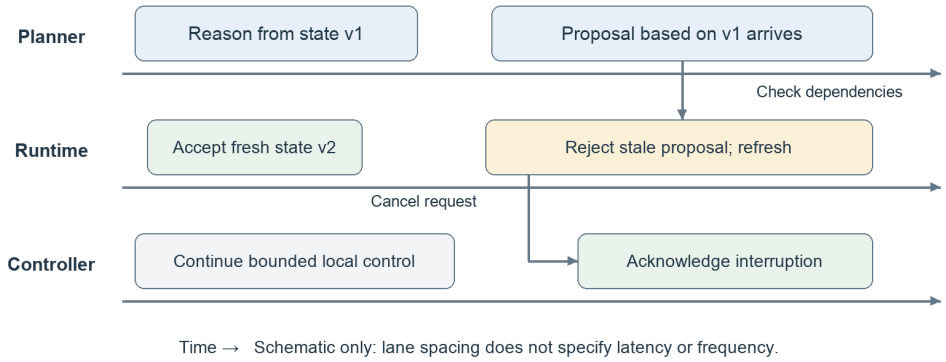

Figure 4. Asynchronous execution can invalidate a proposal before it arrives. A proposal derived from v1 must be checked against updated state v2; cancellation is distinct from acknowledged interruption. Illustrative timeline, not measured timing data [25, 26, 30, 93].

3.3 Contact dynamics and policy handoffs

Contact-rich manipulation couples fast local feedback with slower task decisions. ACT and Diffusion Policy provide learned execution mechanisms [4, 5], while task-and-motion planning reasons about feasibility across symbolic and geometric choices [27, 28]. A harness can combine such components, but successful composition depends on the state produced at their boundaries, not simply on a compatible function signature.

Harness VLA uses a frozen VLA for retryable contact-rich phases and analytic primitives for complementary operations, learning the operating range of this fixed library through execution experience [19]. RoboHarness

instead emphasizes routing among heterogeneous policies and using retrieved experience to make handoffs compatible with the next policy [20]. Both expose a compositional failure mode: individually competent components can fail when one leaves an unsuitable initial state for another. Evaluation should therefore measure transition failures separately from failures within a skill.

Figure 5 separates the exit condition of one skill from the entry condition of the next. The harness can repair the transition through a contracted restaging action, but must not label every such repair as recovery of the underlying policy. This distinction also determines which failure rate should be reported [19, 20].

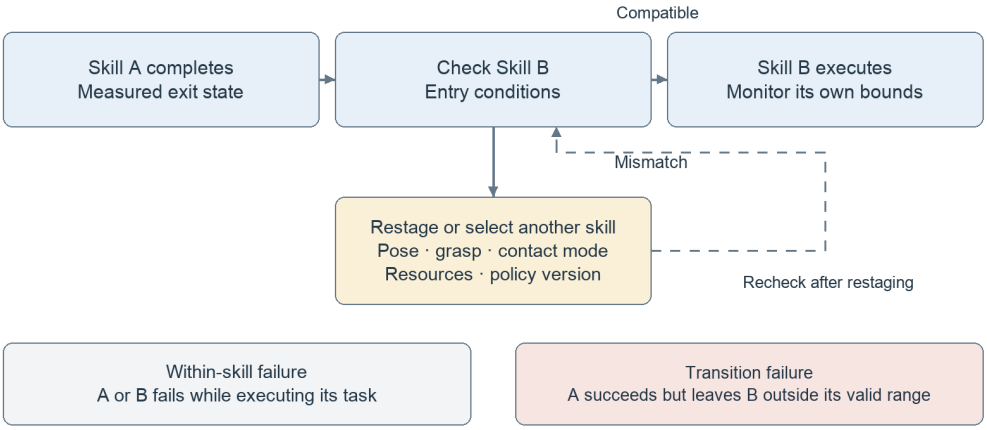

Figure 5. Policy handoff as a compatibility problem. A successful exit from skill A may violate entry conditions of skill B; restaging must itself be verified. Within-skill and transition failures require separate evaluation. Author synthesis [4, 5, 19, 20, 27].

3.4 Distribution shift and embodied uncertainty

Domain and dynamics randomization improve transfer by exposing perception or control models to variability during learning [85, 86]. Runtime mechanisms address a different stage: detecting when current conditions are inconsistent with the assumptions under which a capability was admitted. Relevant signals include unexpected object motion, failed postconditions, repeated retries, sensor disagreement, or changes in robot capability. These signals can trigger a conservative action, a new observation, or a change of plan; they do not imply that the distribution shift has been solved.

Prediction and memory are complementary but fallible resources. A world model extrapolates consequences from learned dynamics; retrieved experience supplies precedents for related conditions. Neither should silently override current contradictory measurements. The architecture should preserve uncertainty and allow abstention when available evidence does not support a physical commitment. Table 3 relates these physical constraints to the corresponding runtime decisions and remaining limitations.

Table 3. Physical constraints mapped to runtime decisions

Constraint	Runtime response	Residual limitation
Irreversibility	Preconditions, monitored execution, bounded recovery	A software rollback cannot undo physical damage [22, 85, 86].
Asynchrony	Timestamped state, admission refresh, resource ownership	Speculative actions can become stale; interrupt latency requires measurement [25, 30].
Contact and handoffs	Local feedback, restaging, transition checks	Compatible APIs do not establish compatible operating distributions [4, 5, 19, 20].
Distribution shift	Detect mismatches, resense, abstain, or replan	Prediction and memory can share incorrect assumptions [6, 7, 85, 86].

Chapter 4 A Unified Taxonomy of Methods

4.1 Classification principles

We classify contributions by the runtime problem they address, using five non-exclusive mechanism families: grounding and executable composition; state and memory; scheduling and coordination; verification and recovery; and adaptation. A system can instantiate several families. This avoids conflating a representation, such as code, with a storage mechanism, such as memory, or an update process, such as self-evolution. Publication date is not a classification dimension.

Four descriptors complement this mechanism view. The execution interface records whether the system invokes skills, generates programs, routes policies, or interprets semantic action units. Intervention timing records whether decisions occur before a skill, during execution, between rollouts, or across deployment sessions. The adaptation target records what changes—context, memory, code, critic, policy weights, or robot configuration. The evidence descriptor records whether the contribution is a conceptual synthesis, a software or simulation study, a physical demonstration, or an assurance result under stated assumptions. These descriptors are analytically separable, but no statistical independence or exhaustive partition is claimed.

A paper is assigned a primary mechanism according to its central runtime contribution and can receive secondary descriptors. Constituent policies and middleware are classified as supporting components, not promoted to complete harnesses. Unreported timings remain unreported; benchmark success is not converted into a safety label. Table 4 applies these rules to representative foundations and integrated systems using the same columns.

Figure 6 provides a reading map for the taxonomy. Mechanism families answer what a contribution does; the four descriptors specify how, when, what changes, and where the evidence applies. A method may appear in several discussions while occupying one row in the comparison table.

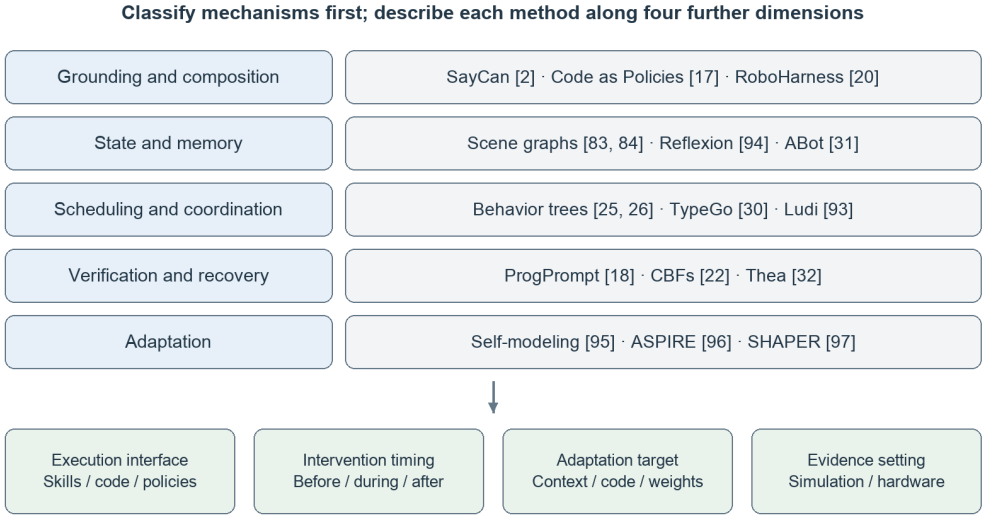

Figure 6. Mechanism families and complementary descriptors. Examples indicate representative contributions, not exclusive memberships or a ranking. Foundations, components, and integrated runtimes must retain their different evidence boundaries [2, 17, 20, 22, 25, 30–32, 83, 93–97].

4.2 Grounding and executable composition

SayCan grounds candidate skills through task relevance and affordance estimates [2]. Code as Policies expresses behavior through generated programs calling perception and control APIs [17], while ProgPrompt incorporates environmental state and assertions in program-like plans [18]. These approaches differ in the

representation used to connect intent and action, not in whether they belong to separate historical generations of harnesses.

The same mechanism family accommodates heterogeneous-policy routing in RoboHarness [20], bounded capability invocation in EmbodiedSkills [79], and semantic action interpretation in Show-Harness [80]. These systems place responsibility at different boundaries: selecting an appropriate executor, specifying an execution lifecycle, or grounding discrete model decisions into local robot actions. A semantic interface can support fine-grained VLM decisions, so "harness" should not be restricted to a slow planner above an otherwise autonomous VLA.

The design tradeoff is between expressive freedom and enforceable execution structure. Generated code offers compositional flexibility but requires restricted APIs and runtime checks. A fixed skill interface narrows behavior but makes preconditions, termination, and resource use easier to specify. An embodiment-specific interpreter provides a stable vocabulary while retaining robot-specific implementation work. The relevant measure is how well the interface supports correct composition under perturbation, not how much code or how many skills it exposes.

Executable programs can also become a reusable skill substrate. ASPIRE combines code-as-policy execution with trace-based diagnosis, repair, validation, and a persistent skill library [96], extending the composition problem illustrated by Code as Policies [17]. RoboClaw couples learned policy orchestration with forward and inverse action pairs for collection and recovery [98]. These approaches differ in what is reused—a programmatic skill or a learned behavior and its recovery partner—but both require explicit conditions for the next invocation.

Multimodal code generation broadens this family beyond text-only program synthesis. RoboCodeX decomposes instructions into object-centric manipulation units within a tree-structured code-generation framework, whereas RoboPro learns policy-code generation from video-derived supervision [99, 100]. CaP-X studies coding agents through a shared manipulation environment and exposes sensitivity to the abstraction level of the available primitives [101]. Together with Code as Policies, these works show that program quality must be interpreted relative to the perception APIs, motion primitives, feedback, and designer knowledge supplied by the execution environment [17].

Guava systematically varies workflow, observation space, and action space, emphasizing iterative perception–reasoning–action, semantic action abstractions, and multimodal observations [102]. This complements Show-Harness and Thea: all expose robot behavior through an interface, but the abstraction and feedback available to the model differ [32, 80]. VLAs-as-Tools adds invocation-aligned policy training and in-execution progress feedback, while Cortex aligns higher-level plans with a canonical skill interface and execution constraints [103, 104]. These systems connect orchestration to the behavior of the executor, so any comparison must disclose both interface design and policy adaptation.

Graph-as-Policy provides a further executable representation: GaP composes perception, planning, and control nodes from a modular skill library, rehearses variants in generated simulation, and refines graph structure and parameters [105]. This extends the composition question from generating a sequence of calls to selecting dependencies and branching behavior. Its reported robotic evaluation differs from generic natural-language harnesses, whose interpreted policies have been assessed on coding, terminal, and computer-use tasks [106]. The transferable idea is an inspectable execution specification; physical validity still requires grounded robot-side evidence.

4.3 State and memory

Scene graphs organize spatial entities and relationships for embodied reasoning [83, 84]. Reflexion illustrates how linguistic feedback can modify subsequent decisions without weight updates [94], although its agent results do not by themselves establish robotic reliability. Within embodied runtimes, graph memory, execution

traces, and explicit session state serve different purposes and should not be collapsed into a single “persistent memory” capability.

ABot-AgentOS emphasizes persistent, source-grounded multimodal memory and isolated execution context [31]. Thea uses scene graphs as context [32], while PhyAgentOS separates explicit session state from cognitive planning [21]. Harness VLA and RoboHarness use experience to inform primitive applicability and policy transitions [19, 20]. Their common question is which remembered evidence is relevant to the next decision, but their stored objects and retrieval targets differ.

Memory introduces a consistency problem as well as a capacity problem. A remembered object location may conflict with a fresh observation; a successful trace may depend on a tool or policy version that has changed. Useful evaluations should perturb stale facts, retract incorrect entries, and test whether provenance permits correction. Improvements from more context should be distinguished from improvements attributable to storage structure, retrieval, or a better underlying perception model.

The conflict-resolution path in Figure 7 connects this representational comparison to execution. Memory can suggest an applicable skill or handoff preparation; current observations must still establish whether the suggestion is relevant. When evidence is insufficient, the runtime should preserve an unknown state and obtain more information rather than silently choosing whichever source is most convenient.

The agent-memory survey by Huang et al. separates memory substrates, cognitive roles, and the subject whose experience is represented [60]. This makes distinctions within embodied memory more precise: a visual keyframe supports episode retrieval, a scene graph supports spatial relations, a successful program supports procedural reuse, and a task ledger supports commitment tracking. Their value depends on the query and decision they serve, not simply on the amount stored.

HELM links episodic keyframe retrieval to a learned state verifier and a controller for rollback and replanning, with evidence on long-horizon manipulation benchmarks [107]. ViReSkill instead stores successful replanned action sequences for subsequent reuse, and the human-in-the-loop framework of Meng et al. turns corrections into reusable skills supported by retrieval [108, 109]. These mechanisms complement ABot-AgentOS and ASPIRE: they differ in whether the reusable unit is an observation, a plan, a corrected program, or a structured memory entry [31, 96]. Learned verification and remembered success remain fallible; rollback must be interpreted according to what the underlying environment can physically restore.

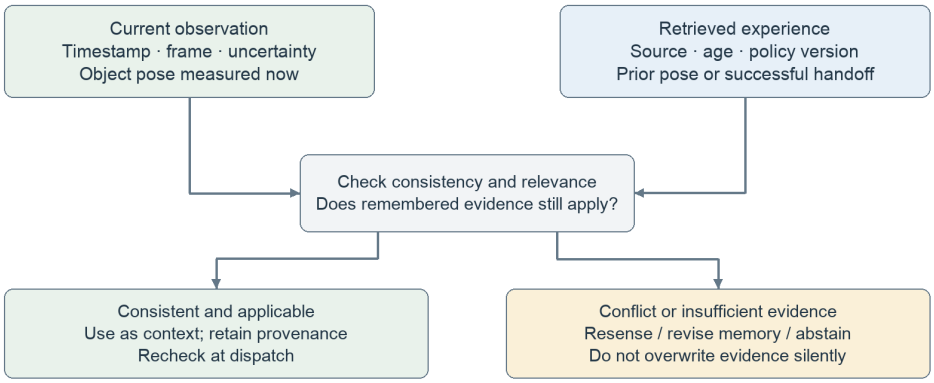

Figure 7. Current observations and persistent experience have different evidentiary roles. Conflict requires an explicit resolution step; a retrieved precedent is not automatically valid in the current scene. Recommended design synthesized from state and memory mechanisms [19–21, 31, 32, 83, 84].

4.4 Scheduling and coordination

Behavior trees provide structured reactive composition [25, 26], whereas an operating-system-style runtime can expose concurrent task processes and resource arbitration [30]. These approaches are not exclusive: a tree

can implement a skill inside a scheduled task. The distinction concerns the unit being coordinated and the semantics of interruption, rather than a presumed low-versus-high control frequency.

Session management in PhyAgentOS [21] and multi-turn interaction in Ludi 0.1 [93] extend coordination beyond sequential task execution. Human clarification, interruption, and goal revision can change the active commitment while the robot is moving. A runtime must reconcile this change with physical state: replacing a conversational goal does not automatically cancel all pending commands or make the current action immediately interruptible.

Multi-agent coordination adds another boundary to the same category. General LLM multi-agent surveys organize role assignment and communication; embodied multi-agent reviews additionally examine perception, planning, communication, and feedback across physical participants [110–112]. GenSwarm provides a concrete code-policy generation and deployment system for multiple robots, while the MARS Challenge separates multi-agent planning and robotic execution in its evaluation setting [113, 114]. Multiple reasoning roles are not equivalent to multiple independently acting robots. A harness must make shared-resource ownership, message freshness, and responsibility for joint failure explicit in either case.

4.5 Verification and recovery

Assertions in ProgPrompt and structured fallback in behavior trees make execution contingencies explicit [18, 25, 26]. Physical barrier-function methods operate at a different level, constraining control under mathematical assumptions [22]. Plan-level LLM verification [24], Thea's outcome evaluation [32], and Edge Skillguard's evidence-dependent invocation authority [33] address distinct propositions. Their inclusion in one mechanism family makes these differences visible rather than implying equivalent assurance.

A complete comparison asks what is checked, when it is checked, who enforces the result, and which failures can escape the check. The same model proposing and judging an action can share errors; a sensor-based predicate can depend on uncertain calibration; a correct permission decision can still admit a physically infeasible action. Recovery evaluation should therefore report false acceptance, false rejection, intervention cost, and residual failure, in addition to eventual task completion.

Verification artifacts also differ in what they establish. Agents4PLC checks generated industrial-control code against specifications, providing an example of executable verification during program construction [115]. FATE audits scene attributes and simulated task feasibility and repairs task or scene proposals, addressing curriculum construction rather than certifying a deployed manipulation plan [116]. Skill-RM organizes heterogeneous reward-evaluation criteria through reusable skills, but its language-agent evaluation results do not establish physical outcome accuracy [117]. These examples motivate recording the proposition checked, the source of evidence, and the point in the lifecycle at which a result can affect behavior.

4.6 Adaptation and deployment support

Adaptation is classified by the artifact that changes. Reflexion updates verbal context [94]; damage-adaptive robotics changes behavior using feedback about altered physical capabilities [95, 118]. SHAPER optimizes reusable skills and external harness structure while keeping the base model frozen [97]. Zetta evolves runtime critics and recovery skills across distinct execution and update loops [119]. These mechanisms are complementary but cannot be summarized by one “self-evolving” label.

RHO optimizes executable neurosymbolic policy repositories through rollout feedback [120], while ASPIRE retains validated programmatic skills discovered through interaction [96]. ENPIRE couples policy improvement with verified physical resets and agent-driven changes to robot learning infrastructure [121]. RoboClaw also links collection, learning, and execution through paired forward and recovery actions [98]. Relative to verbal context updates [94] and harness-artifact optimization [97, 119], these systems make the object of adaptation and the cost of obtaining new embodied experience especially important. They should be

compared by update target and deployment mode, not grouped together solely because they use the term agentic.

Surveys of self-evolution distinguish the object, timing, and mechanism of adaptation, while agent-adaptation research separates changes to the agent from changes to its tools [122, 123]. Xiang et al. extend the discussion to model–environment co-evolution; Jiang et al. emphasize experience becoming memory, skills, and feedback; and Ren et al. study how experience drives controlled changes across agentic systems [124–126]. These perspectives support the adaptation-target descriptor and caution against treating all forms of improvement as weight learning or, conversely, as training-free harness evolution.

SkillsCrafter addresses continual language-conditioned manipulation through mechanisms for preserving and aggregating learned skill knowledge [127]. DreamTeam optimizes external workspace artifacts in an interactive abstract-game environment [128]. These occupy different points in the taxonomy from SHAPER, RHO, and ENPIRE: the changed artifact and the source of feedback differ, even where all methods iterate on experience [97, 120, 121]. A useful comparison therefore asks what persists, how candidates are validated, and whether gains transfer to held-out tasks rather than relying on a shared self-improvement label.

Deployment support requires its own descriptor. SPINE organizes robot profiling, debugging, repair, and validation workflows [129]. This can improve the readiness of the execution stack, but it should not be counted as evidence that a robot can mechanically repair itself during a mission. Likewise, policy adaptation within an orchestration framework must be distinguished from improvement caused by the harness itself [79]. The taxonomy records both the runtime mechanism and the object of change.

Research and deployment harnesses should also be visible in the comparison. HARBOR structures simulation reinforcement-learning workflows with staged execution, persistent artifacts, validation gates, and parallel trials [130]. Nautilus builds adapters and validated workflows across policies, benchmarks, and robot platforms [131]. Articraft supplies a constrained programming interface and validation feedback for articulated asset generation [132]. These systems support robot learning and evaluation rather than necessarily mediating every action during a mission. Their contribution is better measured through reproduction success, engineering effort, artifact validity, and resulting policy performance, with physical deployment assessed separately.

Table 4. Unified comparison by mechanism and execution boundary

Work or foundation	Primary mechanism and interface	Intervention or adaptation target	Evidence boundary
SayCan [2]	Grounding; affordance-weighted skill selection	At skill selection; pretrained skill set	Physical tasks; affordance is not a safety proof.
Task and motion planning [27, 28]	Grounding; symbolic and geometric feasibility	Plan search and refinement	Planning foundation; depends on models and representations.
Code as Policies [17]	Composition; generated programs over APIs	Program structure and parameters	Embodied control; code validity differs from physical validity.
RoboCodeX; RoboPro [99, 100]	Composition; multimodal policy-code generation	Training and program synthesis	Model and runtime both shape the generated behavior.
ProgPrompt [18]	Composition and checks; situated task programs	Plan assertions and contingent execution	Task execution evidence; no universal safety certificate.
Show-Harness [80]	Grounding; discrete semantic action interface	VLM decisions with robot-specific interpretation	Zero-shot and fine-tuned variants; distinguish their evidence.

Work or foundation	Primary mechanism and interface	Intervention or adaptation target	Evidence boundary
CaP-X; Guava [101, 102]	Composition; interfaces and closed-loop feedback	Action abstraction and observation design	Compare scaffolding, interaction budgets, and any model training.
EmbodiedSkills [79]	Composition; contracted skill lifecycle	Orchestration and optional policy adaptation	Separate harness effects from adapted-policy gains.
VLAs-as-Tools; Cortex [103, 104]	Grounding; aligned skill/policy invocation	Policy training, progress feedback, and task transitions	Separate executor alignment from high-level orchestration.
RoboHarness [20]	Composition and memory; heterogeneous policy routing	Capability choice and handoff preparation	Long-horizon execution; transition compatibility is central.
Harness VLA [19]	Composition and memory; frozen VLA plus fixed primitives	Operating-range knowledge and restaging	Perturbed manipulation tasks; no policy weight update.
GaP [105]	Composition; graph-structured robot programs	Graph structure and parameters	Simulation rehearsal and reported physical tasks; library assumptions matter.
RoboClaw [98]	Orchestration and recovery; learned policies with paired actions	Collection, learning, execution, and inverse recovery	Physical tasks; reset feasibility is part of the system.
Scene graphs [83, 84]	State; spatial entities and relations	Environmental representation	Perception foundation, not a complete agent runtime.
ABot-AgentOS [31]	Memory and lifecycle; context-isolated skills	Graph memory and gated runtime assets	Benchmark subset and memory evaluations; not universal reliability.
HELM [107]	Memory, verification, and recovery	Episodic retrieval, learned verifier, replanning	Long-horizon manipulation benchmarks; verifier is learned.
ViReSkill; lifelong code skills [108, 109]	Memory and adaptation; successful/corrected skills	Stored plans or programs	Distinguish autonomous feedback from human corrections.
Thea [32]	State and verification; tools and scene-graph context	Outcome evaluation through exit semantics	Task-outcome evidence differs from physical safety.
Behavior trees [25, 26]	Coordination; reactive control structure	Condition evaluation and fallback	Architectural foundation; behavior depends on leaf implementations.
TypeGo [30]	Scheduling; tasks and typed subsystems	Preemption, arbitration, and speculative planning	Early robot prototype; measured responsiveness is not a hard deadline proof.
PhyAgentOS [21]	Coordination; session-centered execution	Explicit state and execution acceptance	Decoupled planning and execution; no general safety proof.

Work or foundation	Primary mechanism and interface	Intervention or adaptation target	Evidence boundary
AEROS; Harnessing Embodied Agents [81, 82]	Coordination and governance; modular capabilities	Admission, monitoring, and recovery	Reported controlled simulation; policy compliance is scoped.
Ludi 0.1 [93]	Interaction and coordination; dialogue with robot capabilities	Clarification, interruption, and goal revision	Social embodied interaction; explicit model and runtime roles.
GenSwarm [113]	Coordination; multi-robot code policies	Policy generation and deployment	Multiple robots require shared-state and resource checks.
Control barrier functions [22]	Physical assurance; control constraints	Controller-level intervention	Guarantees depend on stated dynamics and feasibility assumptions.
Agentic Harnesses [24]	Verification; LLM plan-review layer	Accept, reject, or escalate before execution	Learned judgment, not formal dynamics verification.
Edge Skillguard [33]	Authority; typed invocation policy	Evidence-dependent permission checks	Edge control-plane testbed; not a general manipulation benchmark.
Reflexion [94]	Memory and adaptation; verbal feedback	Context across trials	General agent mechanism; robotic transfer needs separate evidence.
ASPIRE [96]	Composition and adaptation; code-as-policy skill library	Trace-driven repair and validated reusable skills	Simulation and real-robot evidence; track task and embodiment scope.
SHAPER [97]	Adaptation; skills and external harness	Rollout-driven artifact optimization	Frozen model; environment interaction still required.
RHO [120]	Adaptation; search over executable policy repositories	Policy code and harness structure during optimization	Benchmark-dependent deployment modes; include search cost.
ENPIRE [121]	Adaptation; verified resets and rollout-driven improvement	Policy plus robot learning infrastructure	Real-world loops; distinguish retries from single-trial success.
Zetta [119]	Verification and adaptation; critics and recovery	Action-time checks and gated updates	Simulation task evidence; distinguish execution and update loops.
SPINE [129]	Deployment support; profiling and debugging	Configuration diagnosis, repair, validation	Deployment workflows, not autonomous mechanical self-repair.
HARBOR; Nautilus [130, 131]	Deployment and learning workflow support	Training pipelines, adapters, validation gates	Research workflow results differ from mission-time control.

4.7 What the comparison establishes

The unified matrix shows that memory, generated code, reactive coordination, and adaptation commonly coexist. It also shows that a narrow mechanism can provide stronger evidence for a specific property than a broad runtime provides for overall reliability. A controller-level constraint and a rich memory system answer different questions, so neither should receive a universal superiority score.

Synthesis papers frame the implementation comparison by clarifying model capability, infrastructure, and environmental coupling [8, 29]. The broader survey literature links these distinctions to action representation, predictive modeling, and harness engineering [41, 47, 53, 58]. Their role is conceptual; surveys and roadmaps are not counted as implemented harnesses or additional experimental baselines.

The expanded literature reveals two orthogonal distinctions. First, model-centered methods modify the capability available to the robot, whereas runtime methods modify how a capability is used; hybrid systems do both [40, 58, 103]. Second, an operational harness governs task execution, while a research harness governs data, training, adaptation, or evaluation workflows [130, 131]. These distinctions refine the same mechanism taxonomy rather than introduce a separate category for recently published work. Each comparison should identify the controlled loop before attributing a gain to the harness.

Chapter 5 Implementation and Compositional Design

5.1 Executable representations and restricted interfaces

Behavior trees, finite-state machines, generated programs, and session-based executors are alternative representations of control flow. Behavior trees expose SUCCESS, FAILURE, and RUNNING and support reusable reactive composition [25, 26]. Code as Policies uses program synthesis over available APIs [17]. A review should not assume that all model output is compiled into XML or that a valid program is a physically valid plan.

Software-agent work helps explain why representation matters. OPENDEV organizes terminal tasks through routing, context management, persistent memory, and explicit execution phases [133]. The governance framing of Kim and Hwang emphasizes machine-enforceable constraints on generated artifacts [134], and code-as-harness research connects executable artifacts to reasoning, action, and verification [55]. For a robot, the analogous benefit is inspectability of the program and its dependencies. The stronger physical obligation is to validate that an admissible program invokes capabilities whose assumptions hold in the current world.

An implementation should validate the chosen representation before admission: referenced capabilities must exist; inputs must satisfy type and range requirements; resource claims must be explicit; and execution must have termination and interruption semantics. Syntactic validation cannot determine whether a grasp is feasible or whether a command is authorized. Those decisions require the state, capability contract, and authority checks described in Chapter 2.

For example, “place the mug on the tray” can be decomposed into object grounding, approach, grasp, transport, release, and outcome verification. Each transition requires observable evidence. A successful gripper-close call alone does not establish that the mug is held. If the grasp fails, the appropriate response may be resensing or restaging before another attempt; repeated retries consume a bounded budget. This is an illustrative engineering pattern, not a performance claim about one surveyed system.

5.2 Predictive services and runtime critics

World models offer a way to estimate action consequences without physically executing every candidate [6, 7]. Their usefulness depends on the predicted variables, horizon, uncertainty, and computational cost. A latent representation may reduce inference cost, but a visually plausible rollout or a compact latent state does not necessarily preserve collision geometry, contact force, or rare failure events. No uniform sub-50 ms or sub-20 ms guarantee follows from using a latent model.

A runtime can use prediction to rank candidates, identify likely failure, or request additional sensing. Admission should still be conditioned on current evidence and independent constraints appropriate to the application. Runtime critics provide another source of intervention signals: Zetta explicitly separates action-time governance, rollout-level proposals, and validation-gated updates [119]. Predictive models and critics should be evaluated for calibration, missed hazards, and intervention timing rather than only for average task success.

World-model support can enter at several different points. Video Prediction Policy uses predicted visual representations for action learning, and Cortex 2.0 evaluates candidate futures before acting in reported industrial manipulation tasks [135, 136]. DreamDojo can act as a learned environment for prediction and policy evaluation, while Cosmos Policy couples action, state, and value prediction [71, 77]. These roles require different interfaces: a latent feature supplied to a policy is not the same object as a scored trajectory exposed to the planner.

Table 5 separates representative predictive-action designs by what happens at deployment. The comparison also explains why a requirement that every WAM render future video would exclude latent or training-only predictive approaches. Conversely, removing explicit rollout at inference does not make a learned policy an external harness. The runtime still needs measured feedback, invocation contracts, and a policy for handling predictions that disagree with observations [59, 74, 75].

Table 5. Predictive information and deployment regimes

Representative design	Predictive information	Deployment and harness implication
VPP; mimic-video [69, 135]	Video-derived predictive latent features condition action decoding.	Record the feature/policy boundary; prediction is not an independent outcome check.
DreamZero; Motus [68, 70]	Coupled video/action generation or switchable predictive modes.	State which mode executes and which outputs are exposed to the runtime.
Cosmos Policy; LingBot-VA [71, 72]	Actions coupled to future state/value prediction or causal frame prediction.	Report planning configuration, observed feedback, and inference scheduling.
BagelVLA [76]	Interleaved language reasoning and visual prediction guide action.	Account for computation inside the policy before attributing runtime gains.
VLA-JEPA; Being-H0.7; Fast-WAM [73–75]	Future-informed training; deployment need not render future observations.	Separate training signal from an online predictive service.
DreamDojo; Cortex 2.0 [77, 136]	Action-conditioned simulation or scored candidate futures.	Validate the service for its planning/evaluation role and target robot setting.

The evaluation target should follow the intended use of the prediction. World-model surveys distinguish physical consistency, state understanding, and downstream decision utility from visual fidelity [47, 48]. The agentic-world-modeling perspective extends this reasoning across physical, digital, social, and scientific environments, whose governing constraints differ [137]. For embodied execution, a realistic-looking future is useful only insofar as its action conditioning, uncertainty, and latency support the decision being made. Prediction error and planning benefit should therefore be measured separately.

The choice of simulation infrastructure also shapes what can be tested. Platforms such as Isaac Gym, MuJoCo, and Orbit support controlled experimentation [138–140], but simulated reset and privileged state can make recovery and verification easier than on hardware. Results should state whether the harness receives information unavailable to a deployed robot and whether evaluation includes sensing and actuation delays.

The simulator–world-model distinction also matters. Physical simulators expose an explicit modeled environment, while learned world models approximate transitions from data; surveys emphasize their complementary roles in training and evaluation [141]. Neither source of synthetic experience establishes real-world reliability on its own. Evaluation should disclose contact assumptions, observation access, reset procedures, and whether the same learned model generates both training experience and test outcomes, since this can hide shared modeling errors.

5.3 State freshness and interruption

Asynchronous proposals should carry a state version, creation time, applicable scene assumptions, and an expiry policy. Dispatch can rebind object references, recompute a trajectory, or invalidate the proposal. A simple pose offset is insufficient when the target identity, contact mode, obstacle layout, or resource owner changes. Jacobian-based local correction is meaningful only within its kinematic and control assumptions; it is not a general remedy for delayed reasoning.

A cancellation protocol should distinguish requested cancellation, acknowledged interruption, and verified cessation or transfer of action. The controller may need to complete a bounded segment before pausing. Pending actions associated with an obsolete goal must be invalidated, and resumed tasks must reacquire resources. TypeGo's scheduling design makes resource coordination explicit [30]; interaction-rich systems such as Ludi 0.1 motivate testing interruption under changing human intent [93].

5.4 Embodiment interfaces and portability

A common interface can stabilize the relationship between reasoning and execution without making robot bodies interchangeable. A task-space pose, a semantic action unit, and a learned policy invocation each hide different assumptions about reachability, sensing, end-effectors, and contact. Middleware transports these requests, but embodiment-specific implementations must supply the corresponding behavior [4, 5, 23, 80].

Portability should be decomposed into interface reuse, implementation reuse, and behavioral transfer. Reusing an API across two robots establishes only the first. Replacing a policy behind an unchanged contract requires revalidating its preconditions, timing, and postconditions. A useful cross-embodiment experiment reports which adapters, demonstrations, calibration procedures, and policy parameters changed, instead of reporting a shared interface as evidence of zero-cost transfer.

Human-video pretraining changes the source of transferable knowledge without removing the embodiment interface. Being-H0.5 uses a unified action space and embodiment-aware modeling, while EgoScale studies transfer from large-scale egocentric human data to dexterous manipulation [142, 143]. A harness consuming these capabilities should record the target embodiment, action parameterization, calibration, and adaptation data. Reported transfer within a study is evidence about that recipe and evaluation set, not a guarantee that any robot exposing the same API can execute the same physical skill.

5.5 Governed updates and traceability

Persistent improvement requires a separation between proposing and deploying a change. A trace should record the goal, observation provenance, capability and policy versions, verification decisions, interventions, and outcome. Candidate memory edits, generated skills, or critics can then be tested against both targeted failures and retained tasks. Figure 8 illustrates this update discipline as a design recommendation.

A physical reset is an experimental intervention, not a free database operation. Paired recovery actions in RoboClaw and reset verification in ENPIRE make this dependency explicit [98, 121]. A failed reset changes the initial conditions of the next rollout; excluding its time or human assistance can overstate autonomous improvement. The software promotion loop and the physical reset loop in Figure 8 must therefore be reported separately.

SHAPER and Zetta place adaptation in external execution artifacts [97, 119], while ABot-AgentOS uses gated promotion of runtime assets to later evaluation splits [31]. Their comparison motivates reporting update cost, regression, retained competence, and evaluation separation. Frozen model weights do not imply a fixed system, and training-free adaptation can still require substantial rollouts. A rollback mechanism can restore a software artifact; it cannot undo physical harm incurred during exploration.

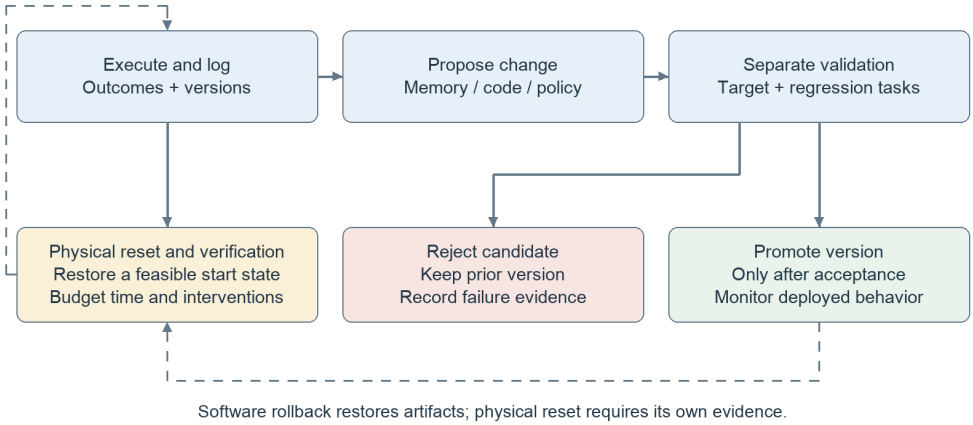

Figure 8. Two coupled loops for improvement: artifact validation and physical reset. Rejected candidates retain the previous software version; restoring the physical start state requires a separately checked action. Recommended synthesis, not a claim that every system implements all steps [31, 96–98, 119–121].

Chapter 6 Evidence and Open Challenges

6.1 Evaluating the contribution of the harness

An end-to-end success rate combines model quality, policy competence, perception, scene difficulty, and runtime design. To attribute a gain to the harness, comparisons should hold these factors constant where possible. Useful ablations remove or replace a specific mechanism—memory retrieval, handoff preparation, outcome verification, or recovery—while preserving task budgets and execution capabilities. If the lower-level policy is also adapted, the result measures a joint system change [79].

Two additional baselines help separate the sources of a gain. CaP-X varies the supplied abstractions and feedback, exposing the contribution of designer scaffolding; HELM compares memory, verification, and recovery components in long-horizon execution [101, 107]. A model–harness comparison should likewise report performance under a fixed model with changing runtime, and a fixed runtime with changing model. Interaction limits, retries, sensor access, training data, and any learned verifier must be accounted for in both conditions.

Evidence settings should remain explicit. TypeGo reports an early physical prototype [30]; Edge Skillguard evaluates authority guards on an edge control-plane testbed [33]; simulation benchmarks in SHAPER and Zetta support claims within their respective task distributions [97, 119]. These results cannot be placed on one numerical leaderboard without aligned tasks, resources, and definitions of success. Table 6 gives a common reporting structure without inventing unreported measurements.

Table 6. Evaluation dimensions and necessary controls

Dimension	Report	Control or diagnostic
Task competence	Success, partial completion, horizon, task diversity	Fix base model, policy, sensors, and task budget.

Dimension	Report	Control or diagnostic
Composition	Handoff failure and entry-condition violations	Separate transition failure from within-skill failure.
Verification	False acceptance, false rejection, unknown outcomes	Use independently grounded outcome labels.
Recovery	Detection delay, recovery success, retry and intervention cost	State recovery budget and human assistance.
Timing	Observation age, tail latency, interrupt acknowledgment	Measure full deployment; do not infer from nominal rates.
Adaptation	Update cost, retained competence, regression, transfer	Separate proposal, validation, and evaluation data.
Authority	Unauthorized admission and revocation behavior	Test adversarial claims and stale authorization.
Physical assurance	Constraint violations and intervention envelope	State sensor, dynamics, controller, and operating assumptions.

Figure 9 makes causal attribution explicit. A comparison with a stronger model, a newly trained policy, or a larger retry budget cannot by itself isolate the harness contribution. Report aligned task budgets and provide a fixed-policy comparison when policy adaptation is also part of the system [79, 121].

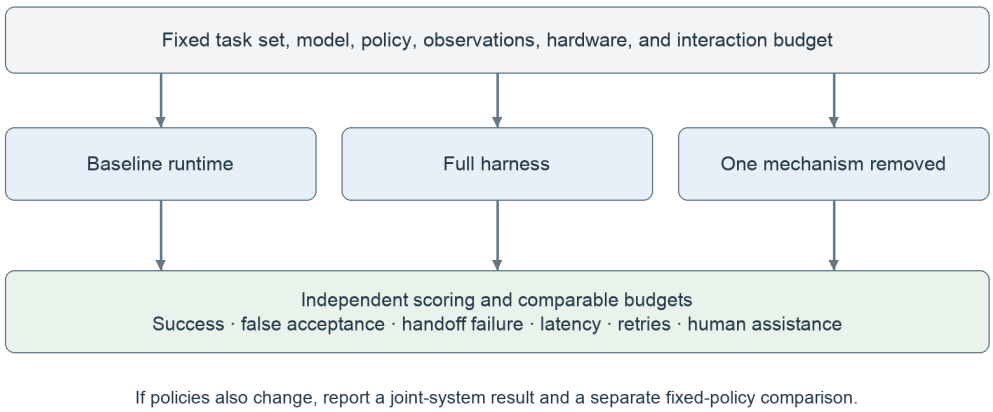

Figure 9. Evaluation that isolates the contribution of a runtime mechanism. Compare aligned baseline, full-system, and ablated conditions with independent outcome scoring. Changes in policy weights or interaction budget require additional controls. Recommended protocol [20, 30, 79, 97, 119, 121].

6.2 Reliability across the full lifecycle

Long-horizon reliability depends on failure detection and containment as well as local competence. Evaluation should include stale context, incorrect memory, interrupted actions, conflicting tasks, policy handoff mismatch, and loss of a required service. A runtime that succeeds after many hidden retries may impose unacceptable time, energy, or contact costs. Tests should therefore expose retry budgets and human intervention rather than absorbing them into an opaque success percentage.

Long horizon should be specified through coupled decisions and state dependencies rather than elapsed time alone, as emphasized by the long-horizon-agent survey [57]. A lengthy repetition of one independent pick-and-place action differs from a sequence in which an early choice constrains later reachability or resources. Skill-evaluation surveys further motivate measuring reuse, transfer, regression, and the effect of evolving libraries,

in addition to episode success [144]. A persistent benchmark should therefore test whether the robot retains valid commitments while discarding stale assumptions across episodes and sessions.

Retry semantics need equal care. The ENPIRE project distinguishes pass@8 performance with in-context retries from single-trial success [121]. A survey should retain that denominator and interaction protocol instead of presenting the number as pass@1 or as independent best-of-eight sampling. More generally, report initial-attempt success, eventual success within the stated budget, total attempts, and reset or human-intervention cost.

Semantic verification also needs an independent reference. When a model both acts and scores its own success, correlated errors can inflate measured reliability. Trace-grounded scoring, held-out human assessment, or independent sensors can help, but each has limitations. “Unknown” should remain an available outcome when observations do not determine whether a postcondition holds. The broader distinction between bounded demonstrations and justified delegation remains central to world-acting systems [29].

6.3 Hardware health and deployment adaptation

Hardware condition can change the feasible operating envelope through temperature, wear, sensor drift, or actuator damage. Work on damage adaptation and continuous self-modeling provides evidence that robots can adjust behavior to altered capabilities under particular experimental conditions [95, 118]. Incorporating such capabilities into an agent runtime requires reliable diagnosis, capability updates, and a decision about when to stop or request maintenance.

Software debugging and mechanical resilience should be separated. SPINE's deployment workflows address profiling and integration problems [129]. A diagnostic repair of a configuration is not equivalent to restoring a damaged actuator. The harness may reduce demand or choose another feasible capability, but “self-healing” should not imply that safe operation is guaranteed after arbitrary hardware degradation. Evaluations should specify the fault class and whether continued operation is permitted.

6.4 Authority and human interaction

A goal, a suggested plan, and permission to actuate are distinct objects. Edge Skillguard treats authority as a typed property checked against state and evidence [33]. LLM-based reviewers can provide an additional accept, reject, or escalation path [24]. Human-facing systems also need clarification and revision mechanisms: Ludi 0.1 links dialogue and embodied behavior in continuing interaction [93]. These contributions should be evaluated together at the authority boundary while retaining their different enforcement mechanisms.

Agent-security surveys identify risks introduced by tool use, mutable memory, autonomy, and communication with external information sources [145, 146]. In an embodied system, retrieved instructions or scene text may enter the same reasoning context as trusted task constraints. The design implication is to preserve their provenance and keep observation content from silently changing execution authority. This addresses an information-flow problem; controller-level protection is still needed for physical constraints that cannot be enforced by a language-agent permission check.

Personalization creates a related persistence problem. Personalized-agent surveys connect user profiles and memory to planning and action [147]. In a shared robot, a remembered preference may inform how a task is performed, but it should not override current consent, resource ownership, or a newly issued stop. Interaction-aware systems such as Ludi 0.1 make dialogue part of the ongoing execution context [93]; evaluating them requires both task outcomes and correct handling of revised or conflicting human instructions.

Physical safety limits are application- and contact-dependent; a single force threshold does not establish compliance with an entire robot-safety standard. This survey therefore does not equate a harness feature with regulatory certification. Traceability can support investigation and accountability, but logs require integrity protection and cannot by themselves guarantee prevention. Authority revocation, policy updates, and human override should be tested under realistic communication and actuator delays.

6.5 Interoperability and research priorities

Interoperability requires agreement on capability semantics as well as message formats. Promising common elements include typed inputs, resource ownership, observation provenance, interruption acknowledgment, postcondition evidence, and versioned traces. These elements should be validated across multiple execution stacks before being presented as a universal embodied operating-system standard. Existing middleware and Agent OS prototypes supply implementation experience, not proof that a single standard has already emerged [21, 23, 30, 31].

The central open problem is compositional assurance: under what assumptions can individually characterized components be combined while retaining useful reliability properties? Progress requires shared failure taxonomies, reproducible perturbations, tests of update regression, and explicit timing and authority contracts. Predictive models may improve planning and risk estimation [8], but their benefits should be demonstrated alongside uncertainty handling and the physical mechanisms that implement intervention.

Navigation and driving provide useful tests of the scope of these conclusions. Foundation-model navigation surveys emphasize embodied planning and grounding, while driving-oriented VLA surveys add tightly coupled motion, real-time decision making, and domain-specific evaluation [148, 149]. The contract and evidence principles can transfer across these domains, but the concrete action bounds, fallback behaviors, and validation requirements differ. Manipulation success should therefore not be presented as evidence of general embodied reliability.

Chapter 7 Conclusion and Research Agenda

7.1 Synthesis

Embodied agentic harnesses organize the boundary between decisions and physical consequences. Across affordance grounding, programmatic control, structured reactive execution, memory-guided orchestration, and evolving critics, the recurring design requirement is an explicit account of state, capability, execution, and feedback [2, 17–20, 25, 119]. These mechanisms are best understood as complementary components rather than mutually exclusive generations of methods.

The architecture in this survey separates logical responsibilities while allowing implementations to distribute them differently. Its contract formulation distinguishes capability invocation from policy internals, and its taxonomy separates the runtime mechanism from intervention timing, adaptation target, and evaluation evidence. This resolves an otherwise misleading comparison between a memory system, a code generator, a low-level controller, and an integrated Agent OS.

Viewed against the broader literature, model progress and harness progress are complementary. LLMs and multimodal models expand interpretation and reasoning; VLA and predictive-action models expand the available behavior and anticipatory representations; the harness structures their use over time [10, 11, 40, 58, 59]. A model may internalize some planning, memory, or verification functions, reducing the external work required, while still needing interfaces to state, resources, and measurable outcomes. This makes the allocation of responsibilities an empirical design choice rather than a fixed hierarchy of model families.

The most consequential distinction is between deciding that an action is useful, deciding that it is authorized, observing that it succeeded, and establishing that it satisfies a physical safety condition. Progress in one does not automatically establish the others [22, 24, 32, 33]. A reliable embodied system must connect these decisions through enforceable interfaces and explicitly state the assumptions that remain outside the harness.

7.2 Research agenda

A practical agenda begins with reproducible execution contracts and shared evaluation protocols. Capability interfaces should expose preconditions, resources, bounded execution, interruption, and outcome evidence. Benchmarks should then isolate which runtime mechanism contributes to performance, test stale-state and

handoff failures, and report intervention cost alongside success. This makes comparisons across planning, memory, scheduling, and verification systems more meaningful [2, 18, 20, 30, 79].

Adaptation requires a second line of evidence: improvements should survive held-out tasks, preserve prior competence, and respect authority and physical constraints. External memory and program updates can improve a frozen model's behavior, but they still change the deployed system and require validation [31, 94, 96, 97, 119, 120]. Verified resets and recovery partners further determine whether repeated physical learning can proceed without hidden human support [98, 121]. Hardware-aware operation and cross-embodiment reuse should similarly be judged by explicit transfer and fault conditions rather than architectural labels [80, 95, 118, 129].

A further priority is to study the interface between predictive learning and runtime assurance. Future-aware representations can improve policies even when no future is rendered at deployment [73–75]; explicit predictive services can support planning and evaluation when they are sufficiently calibrated and timely [71, 77]. Comparisons should determine which errors require better learned capabilities and which require better admission, state management, or recovery. This connects the WAM research agenda to compositional execution without treating model prediction as a substitute for observed evidence.

The field's long-term objective is sustained, accountable physical autonomy. The reviewed work supports concrete advances in grounding, orchestration, and feedback, while leaving open the reliability of their composition in changing environments. Treating the harness as a measurable set of execution responsibilities provides a basis for that progress without assuming that richer models, more memory, or additional generated skills alone resolve the physical and organizational limits of delegation [8, 29].

References

[1] Brohan, A., et al. (2023). RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. arXiv:2307.15818. https://arxiv.org/abs/2307.15818

[2] Ahn, M., et al. (2022). Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. arXiv:2204.01691. https://arxiv.org/abs/2204.01691

[3] Driess, D., et al. (2023). PaLM-E: An Embodied Multimodal Language Model. International Conference on Machine Learning. https://arxiv.org/abs/2303.03378

[4] Zhao, T. Z., Kumar, V., Levine, S., & Finn, C. (2023). Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. Robotics: Science and Systems. https://arxiv.org/abs/2304.13705

[5] Chi, C., et al. (2023). Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. Robotics: Science and Systems. https://arxiv.org/abs/2303.04137

[6] Ha, D., & Schmidhuber, J. (2018). Recurrent World Models Facilitate Policy Evolution. Advances in Neural Information Processing Systems. https://proceedings.neurips.cc/paper/2018/hash/2de5d16682c3c35007e4e92982f1a2ba-Abstract.html

[7] Hafner, D., et al. (2023). Mastering Diverse Domains through World Models. arXiv:2301.04104. https://arxiv.org/abs/2301.04104

[8] Liang, Y., et al. (2026). From World Action Models to Embodied Brains: A Roadmap for Open-World Physical Intelligence. arXiv:2607.11689. https://arxiv.org/abs/2607.11689

[9] Bommasani, R., et al. (2021). On the Opportunities and Risks of Foundation Models. arXiv preprint arXiv:2108.07258v3. https://arxiv.org/abs/2108.07258v3

[10] Zhao, W. X., et al. (2023). A Survey of Large Language Models. arXiv preprint arXiv:2303.18223v19. https://arxiv.org/abs/2303.18223v19

[11] Li, C., et al. (2023). Multimodal Foundation Models: From Specialists to General-Purpose Assistants. arXiv preprint arXiv:2309.10020v1. https://arxiv.org/abs/2309.10020v1

[12] Xie, J., et al. (2024). Large Multimodal Agents: A Survey. arXiv preprint arXiv:2402.15116v1. https://arxiv.org/abs/2402.15116v1

[13] Wang, L., et al. (2023). A Survey on Large Language Model based Autonomous Agents. arXiv preprint arXiv:2308.11432v7. https://arxiv.org/abs/2308.11432v7

[14] Xi, Z., et al. (2023). The Rise and Potential of Large Language Model Based Agents: A Survey. arXiv preprint arXiv:2309.07864v3. https://arxiv.org/abs/2309.07864v3

[15] Liu, Y., et al. (2024). Aligning Cyber Space with Physical World: A Comprehensive Survey on Embodied AI. arXiv preprint arXiv:2407.06886v8. https://arxiv.org/abs/2407.06886v8

[16] Feng, T., et al. (2025). Embodied AI: From LLMs to World Models. arXiv preprint arXiv:2509.20021v1. https://arxiv.org/abs/2509.20021v1

[17] Liang, J., et al. (2023). Code as Policies: Language Model Programs for Embodied Control. IEEE International Conference on Robotics and Automation. https://arxiv.org/abs/2209.07753

[18] Singh, I., et al. (2023). ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. IEEE International Conference on Robotics and Automation. https://arxiv.org/abs/2209.11302

[19] Zhang, Y., et al. (2026). Harness VLA: Steering Frozen VLAs into Reliable Manipulation Primitives via Memory-Guided Agents. arXiv:2607.08448. https://arxiv.org/abs/2607.08448

[20] Huang, J., et al. (2026). RoboHarness: Memory-Driven Orchestration of Heterogeneous Robot Policies for Long-Horizon Planning. arXiv:2607.18060. https://arxiv.org/abs/2607.18060

[21] Liu, Y., et al. (2026). PhyAgentOS: A Self-Evolving Operating System for Embodied Agents with Decoupled Cognitive Planning and Physical Execution. arXiv:2607.16636. https://arxiv.org/abs/2607.16636

[22] Ames, A. D., Coogan, S., Egerstedt, M., Notomista, G., Sreenath, K., & Tabuada, P. (2019). Control Barrier Functions: Theory and Applications. European Control Conference. https://arxiv.org/abs/1903.11199

[23] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. Science Robotics, 7, eabm6074. https://doi.org/10.1126/scirobotics.abm6074

[24] Bhagra, R., Halapannavar, M., & Bhattarai, U. (2026). Agentic Harnesses: LLM-Driven Verification Layers for Robot Autonomy. arXiv:2608.09857. https://arxiv.org/abs/2608.09857

[25] Colledanchise, M., & Ögren, P. (2018). Behavior Trees in Robotics and AI: An Introduction. CRC Press. https://doi.org/10.1201/9780429489105

[26] Iovino, M., Scukins, E., Styrud, J., Ögren, P., & Smith, C. (2022). A survey of Behavior Trees in robotics and AI. Robotics and Autonomous Systems, 154, 104096. https://doi.org/10.1016/j.robot.2022.104096

[27] Kaelbling, L. P., & Lozano-Pérez, T. (2011). Hierarchical Task and Motion Planning in the Now. IEEE International Conference on Robotics and Automation, 1470–1477. https://people.csail.mit.edu/tlp/pdf/2011/hpnICRA11Final.pdf

[28] Garrett, C. R., et al. (2021). Integrated Task and Motion Planning. Annual Review of Control, Robotics, and Autonomous Systems, 4. https://arxiv.org/abs/2010.01083

[29] Zhu, L., & Cai, M. (2026). From Language Models to World-Acting Systems: Progress and Limits of Agentic AI across Digital, Social, Virtual, and Physical Environments. arXiv:2609.04894. https://arxiv.org/abs/2609.04894

[30] Chen, G., Schott, A., & Zhong, L. (2026). TypeGo: An OS Runtime for Embodied Agents. arXiv:2607.05482. https://arxiv.org/abs/2607.05482

[31] Tian, J., et al. (2026). ABot-AgentOS: A General Robotic Agent OS with Lifelong Multi-modal Memory. arXiv:2607.10350. https://arxiv.org/abs/2607.10350

[32] Wang, Q., et al. (2026). Towards the Harness of Embodied Agents. arXiv:2608.11246. https://arxiv.org/abs/2608.11246

[33] Zhan, Z., & Haddadi, H. (2026). Auto-Policy, not Auto-Skill: Compiled Agent Skills for the Physical World. arXiv:2608.25091. https://arxiv.org/abs/2608.25091

[34] Zhang, Z., et al. (2023). Igniting Language Intelligence: The Hitchhiker's Guide From Chain-of-Thought Reasoning to Language Agents. arXiv preprint arXiv:2311.11797v1. https://arxiv.org/abs/2311.11797v1

[35] Wei, T., et al. (2026). Agentic Reasoning for Large Language Models. arXiv preprint arXiv:2601.12538v1. https://arxiv.org/abs/2601.12538v1

[36] Li, X. (2024). A Review of Prominent Paradigms for LLM-Based Agents: Tool Use (Including RAG), Planning, and Feedback Learning. arXiv preprint arXiv:2406.05804v6. https://arxiv.org/abs/2406.05804v6

[37] Luo, J., et al. (2025). Large Language Model Agent: A Survey on Methodology, Applications and Challenges. arXiv preprint arXiv:2503.21460v1. https://arxiv.org/abs/2503.21460v1

[38] Huang, Y. (2024). Levels of AI Agents: from Rules to Large Language Models. arXiv preprint arXiv:2405.06643v2. https://arxiv.org/abs/2405.06643v2

[39] Xing, E., et al. (2026). Critique of Agent Model. arXiv preprint arXiv:2606.23991v1. https://arxiv.org/abs/2606.23991v1

[40] Ma, Y., et al. (2024). A Survey on Vision-Language-Action Models for Embodied AI. arXiv preprint arXiv:2405.14093v8. https://arxiv.org/abs/2405.14093v8

[41] Zhong, Y., et al. (2025). A Survey on Vision-Language-Action Models: An Action Tokenization Perspective. arXiv preprint arXiv:2507.01925v1. https://arxiv.org/abs/2507.01925v1

[42] Shao, R., et al. (2025). Large VLM-based Vision-Language-Action Models for Robotic Manipulation: A Survey. arXiv preprint arXiv:2508.13073v3. https://arxiv.org/abs/2508.13073v3

[43] Zhang, D., et al. (2025). Pure Vision Language Action (VLA) Models: A Comprehensive Survey. arXiv preprint arXiv:2509.19012v3. https://arxiv.org/abs/2509.19012v3

[44] Kawaharazuka, K., et al. (2025). Vision-Language-Action Models for Robotics: A Review Towards Real-World Applications. arXiv preprint arXiv:2510.07077v1. https://arxiv.org/abs/2510.07077v1

[45] Zhu, Z., et al. (2024). Is Sora a World Simulator? A Comprehensive Survey on General World Models and Beyond. arXiv preprint arXiv:2405.03520v2. https://arxiv.org/abs/2405.03520v2

[46] Ding, J., et al. (2024). Understanding World or Predicting Future? A Comprehensive Survey of World Models. arXiv preprint arXiv:2411.14499v4. https://arxiv.org/abs/2411.14499v4

[47] Li, X., et al. (2025). A Comprehensive Survey on World Models for Embodied AI. arXiv preprint arXiv:2510.16732v3. https://arxiv.org/abs/2510.16732v3

[48] Hou, B., et al. (2026). World Model for Robot Learning: A Comprehensive Survey. arXiv preprint arXiv:2605.00080v1. https://arxiv.org/abs/2605.00080v1

[49] Sun, J., et al. (2025). Integrating World Models into Vision Language Action and Navigation: A Comprehensive Survey. TechRxiv preprint. https://doi.org/10.36227/techrxiv.176531987.77979037/v1

[50] Tan, W., et al. (2026). Towards Generalist Embodied AI: A Survey on World Models for VLA Agents. TechRxiv preprint. https://doi.org/10.36227/techrxiv.176948355.54623875/v1

[51] He, C., et al. (2026). Harness Engineering for Language Agents: The Harness Layer as Control, Agency, and Runtime. Preprints, version 2. https://doi.org/10.20944/preprints202603.1756.v2

[52] Meng, Q., et al. (2026). Agent Harness for Large Language Model Agents: A Survey. Preprints, version 3. https://doi.org/10.20944/preprints202604.0428.v3

[53] Li, J., et al. (2026). Agent Harness Engineering: A Survey. Preprint, OpenReview Archive. https://openreview.net/pdf?id=eONq7FdiHa

[54] Zhou, C., et al. (2026). Externalization in LLM Agents: A Unified Review of Memory, Skills, Protocols and Harness Engineering. arXiv preprint arXiv:2604.08224v1. https://arxiv.org/abs/2604.08224v1

[55] Ning, X., et al. (2026). Code as Agent Harness. arXiv preprint arXiv:2605.18747v1. https://arxiv.org/abs/2605.18747v1

[56] Guo, J., et al. (2026). From Question Answering to Task Completion: A Survey on Agent System and Harness Design. arXiv preprint arXiv:2606.20683v1. https://arxiv.org/abs/2606.20683v1

[57] Dong, G., et al. (2026). Towards Long-Horizon Agents: A Survey. Preprints, version 1. https://doi.org/10.20944/preprints202607.1328.v1

[58] Wang, S., et al. (2026). World Action Models: The Next Frontier in Embodied AI. arXiv preprint arXiv:2605.12090v1. https://arxiv.org/abs/2605.12090v1

[59] Shen, Q., et al. (2026). World Action Models: A Survey. arXiv preprint arXiv:2606.20781v1. https://arxiv.org/abs/2606.20781v1

[60] Huang, W., et al. (2026). A Survey of Agent Memory in the Second Half: Towards Self-Evolving and Long-Horizon Agents. arXiv preprint arXiv:2602.06052v4. https://arxiv.org/abs/2602.06052v4

[61] Xu, R., et al. (2026). Agent Skills for Large Language Models: Architecture, Acquisition, Security, and the Path Forward. arXiv preprint arXiv:2602.12430v4. https://arxiv.org/abs/2602.12430v4

[62] Ding, T., et al. (2026). Always-On Agents: A Survey of Persistent Memory, State, and Governance in LLM Agents. arXiv preprint arXiv:2606.30306v1. https://arxiv.org/abs/2606.30306v1

[63] Black, K., et al. (2024). π0: A Vision-Language-Action Flow Model for General Robot Control. arXiv preprint arXiv:2410.24164v4. https://arxiv.org/abs/2410.24164v4

[64] Physical Intelligence, et al. (2025). π0.5: a Vision-Language-Action Model with Open-World Generalization. arXiv preprint arXiv:2504.16054v1. https://arxiv.org/abs/2504.16054v1

[65] Physical Intelligence, et al. (2026). π0.7: a Steerable Generalist Robotic Foundation Model with Emergent Capabilities. arXiv preprint arXiv:2604.15483v2. https://arxiv.org/abs/2604.15483v2

[66] Xiang, T., et al. (2025). Parallels Between VLA Model Post-Training and Human Motor Learning: Progress, Challenges, and Trends. arXiv preprint arXiv:2506.20966v2. https://arxiv.org/abs/2506.20966v2

[67] Deng, H., et al. (2025). A Survey on Reinforcement Learning of Vision-Language-Action Models for Robotic Manipulation. TechRxiv preprint. https://doi.org/10.36227/techrxiv.176531955.54563920/v1

[68] Bi, H., et al. (2025). Motus: A Unified Latent Action World Model. arXiv preprint arXiv:2512.13030v2. https://arxiv.org/abs/2512.13030v2

[69] Pai, J., et al. (2025). mimic-video: Video-Action Models for Generalizable Robot Control Beyond VLAs. arXiv preprint arXiv:2512.15692v2. https://arxiv.org/abs/2512.15692v2

[70] Ye, S., et al. (2026). World Action Models are Zero-shot Policies. arXiv preprint arXiv:2602.15922v1. https://arxiv.org/abs/2602.15922v1

[71] Kim, M. J., et al. (2026). Cosmos Policy: Fine-Tuning Video Models for Visuomotor Control and Planning. arXiv preprint arXiv:2601.16163v1. https://arxiv.org/abs/2601.16163v1

[72] Li, L., et al. (2026). Causal World Modeling for Robot Control. arXiv preprint arXiv:2601.21998v2. https://arxiv.org/abs/2601.21998v2

[73] Sun, J., et al. (2026). VLA-JEPA: Enhancing Vision-Language-Action Model with Latent World Model. arXiv preprint arXiv:2602.10098v2. https://arxiv.org/abs/2602.10098v2

[74] Luo, H., et al. (2026). Being-H0.7: A Latent World-Action Model from Egocentric Videos. arXiv preprint arXiv:2605.00078v1. https://arxiv.org/abs/2605.00078v1

[75] Yuan, T., et al. (2026). Fast-WAM: Do World Action Models Need Test-time Future Imagination?. arXiv preprint arXiv:2603.16666v2. https://arxiv.org/abs/2603.16666v2

[76] Hu, Y., et al. (2026). BagelVLA: Enhancing Long-Horizon Manipulation via Interleaved Vision-Language-Action Generation. arXiv preprint arXiv:2602.09849v2. https://arxiv.org/abs/2602.09849v2

[77] Gao, S., et al. (2026). DreamDojo: A Generalist Robot World Model from Large-Scale Human Videos. arXiv preprint arXiv:2602.06949v1. https://arxiv.org/abs/2602.06949v1

[78] Lee, S., et al. (2026). Harness Engineering for Physical AI: Robot Middleware Is the Harness Layer. arXiv preprint arXiv:2606.09416v1. https://arxiv.org/abs/2606.09416v1

[79] Wang, W., et al. (2026). EmbodiedSkills: A Unified Framework for Orchestrating, Training, and Deploying VLA Agents. arXiv:2609.01281. https://arxiv.org/abs/2609.01281

[80] Chen, Y., et al. (2026). Show-Harness: Just a VLM Agent Can Play Robots. arXiv:2609.10522. https://arxiv.org/abs/2609.10522

[81] Qin, X., et al. (2026). AEROS: A Single-Agent Operating Architecture with Embodied Capability Modules. arXiv preprint arXiv:2604.07039v3. https://arxiv.org/abs/2604.07039v3

[82] Qin, X., et al. (2026). Harnessing Embodied Agents: Runtime Governance for Policy-Constrained Execution. arXiv preprint arXiv:2604.07833v4. https://arxiv.org/abs/2604.07833v4

[83] Rosinol, A., Gupta, A., Abate, M., Shi, J., & Carlone, L. (2020). 3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans. Robotics: Science and Systems. https://arxiv.org/abs/2002.06289

[84] Hughes, N., Chang, Y., & Carlone, L. (2022). Hydra: A Real-time Spatial Perception System for 3D Scene Graph Construction and Optimization. Robotics: Science and Systems. https://arxiv.org/abs/2201.13360

[85] Tobin, J., et al. (2017). Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. IEEE/RSJ International Conference on Intelligent Robots and Systems. https://arxiv.org/abs/1703.06907

[86] Peng, X. B., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2018). Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. IEEE International Conference on Robotics and Automation. https://arxiv.org/abs/1710.06537

[87] Zheng, Y., et al. (2024). A Survey of Embodied Learning for Object-Centric Robotic Manipulation. arXiv preprint arXiv:2408.11537v1. https://arxiv.org/abs/2408.11537v1

[88] Bai, S., et al. (2025). Towards a Unified Understanding of Robot Manipulation: A Comprehensive Survey. arXiv preprint arXiv:2510.10903v2. https://arxiv.org/abs/2510.10903v2

[89] Cao, Y., et al. (2025). Reconstructing 4D Spatial Intelligence: A Survey. arXiv preprint arXiv:2507.21045v2. https://arxiv.org/abs/2507.21045v2

[90] Zheng, X., et al. (2025). Multimodal Spatial Reasoning in the Large Model Era: A Survey and Benchmarks. arXiv preprint arXiv:2510.25760v2. https://arxiv.org/abs/2510.25760v2

[91] Felicia, G., et al. (2026). From Perception to Action: Spatial AI Agents and World Models. arXiv preprint arXiv:2602.01644v1. https://arxiv.org/abs/2602.01644v1

[92] Yu, S., et al. (2025). How Far are VLMs from Visual Spatial Intelligence? A Benchmark-Driven Perspective. arXiv preprint arXiv:2509.18905v2. https://arxiv.org/abs/2509.18905v2

[93] Chung, W., et al. (2026). Ludi0.1: An Agentic System for Socially Intelligent Robots. arXiv:2608.22035. https://arxiv.org/abs/2608.22035

[94] Shinn, N., et al. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. Advances in Neural Information Processing Systems. https://arxiv.org/abs/2303.11366

[95] Bongard, J., Zykov, V., & Lipson, H. (2006). Resilient Machines Through Continuous Self-Modeling. Science, 314, 1118–1121. https://doi.org/10.1126/science.1133687

[96] Lu, R., et al. (2026). ASPIRE: Agentic /Skills Discovery for Robotics. arXiv:2607.00272. https://arxiv.org/abs/2607.00272

[97] Wang, P., et al. (2026). Self-Evolving Embodied Agents via Skill-Harness Evolution. arXiv:2608.11350. https://arxiv.org/abs/2608.11350

[98] Li, R., et al. (2026). RoboClaw: An Agentic Framework for Scalable Long-Horizon Robotic Tasks. arXiv:2603.11558. https://arxiv.org/abs/2603.11558

[99] Mu, Y., et al. (2024). RoboCodeX: Multimodal Code Generation for Robotic Behavior Synthesis. arXiv preprint arXiv:2402.16117v1. https://arxiv.org/abs/2402.16117v1

[100] Xie, S., et al. (2025). Robotic Programmer: Video Instructed Policy Code Generation for Robotic Manipulation. arXiv preprint arXiv:2501.04268v1. https://arxiv.org/abs/2501.04268v1

[101] Fu, L., et al. (2026). CaP-X: A Framework for Benchmarking and Improving Coding Agents for Robot Manipulation. arXiv preprint arXiv:2603.22435v2. https://arxiv.org/abs/2603.22435v2

[102] Liu, H., et al. (2026). Guava: An Effective and Universal Harness for Embodied Manipulation. arXiv preprint arXiv:2606.18363v1. https://arxiv.org/abs/2606.18363v1

[103] Lei, Z., et al. (2026). Towards Long-horizon Embodied Agents with Tool-Aligned Vision-Language-Action Models. arXiv preprint arXiv:2605.13119v1. https://arxiv.org/abs/2605.13119v1

[104] Peng, J., et al. (2026). Cortex: A Bidirectionally Aligned Embodied Agent Framework for Long-horizon Manipulation. arXiv preprint arXiv:2607.05377v1. https://arxiv.org/abs/2607.05377v1

[105] Chen, K., et al. (2026). GaP: A Graph-as-Policy Multi-Agent Self-Learning Harness For Variational Automation Tasks. arXiv preprint arXiv:2607.05369v1. https://arxiv.org/abs/2607.05369v1

[106] Pan, L., et al. (2026). Natural-Language Agent Harnesses. arXiv preprint arXiv:2603.25723v2. https://arxiv.org/abs/2603.25723v2

[107] Zeng, Z., et al. (2026). HELM: Harness-Enhanced Long-horizon Memory for Vision-Language-Action Manipulation. arXiv preprint arXiv:2604.18791v1. https://arxiv.org/abs/2604.18791v1

[108] Kagaya, T., et al. (2025). ViReSkill: Vision-Grounded Replanning with Skill Memory for LLM-Based Planning in Lifelong Robot Learning. arXiv preprint arXiv:2509.24219v1. https://arxiv.org/abs/2509.24219v1

[109] Meng, Y., et al. (2025). Growing with Your Embodied Agent: A Human-in-the-Loop Lifelong Code Generation Framework for Long-Horizon Manipulation Skills. arXiv preprint arXiv:2509.18597v2. https://arxiv.org/abs/2509.18597v2

[110] Guo, T., et al. (2024). Large Language Model based Multi-Agents: A Survey of Progress and Challenges. arXiv preprint arXiv:2402.01680v2. https://arxiv.org/abs/2402.01680v2

[111] Wu, D., et al. (2025). Generative Multi-Agent Collaboration in Embodied AI: A Systematic Review. arXiv preprint arXiv:2502.11518v1. https://arxiv.org/abs/2502.11518v1

[112] Feng, Z., et al. (2025). Multi-agent Embodied AI: Advances and Future Directions. arXiv preprint arXiv:2505.05108v2. https://arxiv.org/abs/2505.05108v2

[113] Ji, W., et al. (2025). GenSwarm: Scalable Multi-Robot Code-Policy Generation and Deployment via Language Models. arXiv preprint arXiv:2503.23875v2. https://arxiv.org/abs/2503.23875v2

[114] Kang, L., et al. (2026). Advances and Innovations in the Multi-Agent Robotic System (MARS) Challenge. arXiv preprint arXiv:2601.18733v1. https://arxiv.org/abs/2601.18733v1

[115] Liu, Z., et al. (2024). Agents4PLC: Automating Closed-loop PLC Code Generation and Verification in Industrial Control Systems using LLM-based Agents. arXiv preprint arXiv:2410.14209v2. https://arxiv.org/abs/2410.14209v2

[116] Wei, B., et al. (2026). FATE: Closed-Loop Feasibility-Aware Task Generation with Active Repair for Physically Grounded Robotic Curricula. arXiv preprint arXiv:2603.01505v1. https://arxiv.org/abs/2603.01505v1

[117] Chen, T., et al. (2026). Skill-RM: Unifying Heterogeneous Evaluation Criteria via Agent Skill. arXiv preprint arXiv:2606.03980v1. https://arxiv.org/abs/2606.03980v1

[118] Cully, A., Clune, J., Tarapore, D., & Mouret, J.-B. (2015). Robots that can adapt like animals. Nature, 521, 503–507. https://doi.org/10.1038/nature14422

[119] Ding, X., et al. (2026). Zetta ζ: An Efficient Closed-Loop Embodied Harness for Self-Evolving Physical Intelligence. arXiv:2608.16590. https://arxiv.org/abs/2608.16590

[120] Elmaaroufi, K., et al. (2026). RHO: Your Coding Agent is Secretly a Roboticist. arXiv:2606.16458. https://arxiv.org/abs/2606.16458

[121] Xiao, W., et al. (2026). ENPIRE: Agentic Robot Policy Self-Improvement in the Real World. arXiv:2606.19980. https://arxiv.org/abs/2606.19980

[122] Gao, H., et al. (2025). A Survey of Self-Evolving Agents: What, When, How, and Where to Evolve on the Path to Artificial Super Intelligence. arXiv preprint arXiv:2507.21046v4. https://arxiv.org/abs/2507.21046v4

[123] Jiang, P., et al. (2025). Adaptation of Agentic AI: A Survey of Post-Training, Memory, and Skills. arXiv preprint arXiv:2512.16301v3. https://arxiv.org/abs/2512.16301v3

[124] Xiang, Z., et al. (2026). A Systematic Survey of Self-Evolving Agents: From Model-Centric to Environment-Driven Co-Evolution. TechRxiv preprint. https://doi.org/10.36227/techrxiv.177203250.05832634/v2

[125] Jiang, C., et al. (2026). Self-Improving Agents in the Era of Experience: A Survey of Self- to Meta-Evolution. Preprint, OpenReview Archive. https://openreview.net/pdf?id=IUltZSgLMm

[126] Ren, Z., et al. (2026). Self-Improvements in Modern Agentic Systems: A Survey. arXiv preprint arXiv:2607.13104v1. https://arxiv.org/abs/2607.13104v1

[127] Wang, X., et al. (2026). Lifelong Language-Conditioned Robotic Manipulation Learning. arXiv preprint arXiv:2603.05160v1. https://arxiv.org/abs/2603.05160v1

[128] Sarafian, E., et al. (2026). Workspace Optimization: How to Train Your Agent. arXiv preprint arXiv:2605.09650v1. https://arxiv.org/abs/2605.09650v1

[129] Ham, M., et al. (2026). SPINE: Bridging the Cyber-Physical Gap with Agentic AI. arXiv:2607.13049. https://arxiv.org/abs/2607.13049

[130] Li, Z., et al. (2026). HARBOR: A Harness Framework for Agentic Robot Reinforcement Learning. arXiv preprint arXiv:2606.08610v1. https://arxiv.org/abs/2606.08610v1

[131] Jin, Y., et al. (2026). Nautilus: From One Prompt to Plug-and-Play Robot Learning. arXiv preprint arXiv:2605.11665v2. https://arxiv.org/abs/2605.11665v2

[132] Zhou, M., et al. (2026). Articraft: An Agentic System for Scalable Articulated 3D Asset Generation. arXiv preprint arXiv:2605.15187v1. https://arxiv.org/abs/2605.15187v1

[133] Bui, N. D. Q. (2026). Building Effective AI Coding Agents for the Terminal: Scaffolding, Harness, Context Engineering, and Lessons Learned. arXiv preprint arXiv:2603.05344v3. https://arxiv.org/abs/2603.05344v3

[134] Kim, J., & Hwang, H. (2026). Harness Engineering: A Governance Framework for AI-Driven Software Engineering. SSRN preprint. https://papers.ssrn.com/sol3/Delivery.cfm/6372119.pdf?abstractid=6372119&mirid=1

[135] Hu, Y., et al. (2024). Video Prediction Policy: A Generalist Robot Policy with Predictive Visual Representations. arXiv preprint arXiv:2412.14803v2. https://arxiv.org/abs/2412.14803v2

[136] Aida, A., et al. (2026). Cortex 2.0: Grounding World Models in Real-World Industrial Deployment. arXiv preprint arXiv:2604.20246v1. https://arxiv.org/abs/2604.20246v1

[137] Chu, M., et al. (2026). Agentic World Modeling: Foundations, Capabilities, Laws, and Beyond. arXiv preprint arXiv:2604.22748v3. https://arxiv.org/abs/2604.22748v3

[138] Makoviychuk, V., et al. (2021). Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning. arXiv:2108.10470. https://arxiv.org/abs/2108.10470

[139] Todorov, E., Erez, T., & Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. IEEE/RSJ International Conference on Intelligent Robots and Systems, 5026–5033. https://doi.org/10.1109/IROS.2012.6386109

[140] Mittal, M., et al. (2023). Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments. IEEE Robotics and Automation Letters, 8(6). https://doi.org/10.1109/LRA.2023.3270034

[141] Long, X., et al. (2025). A Survey: Learning Embodied Intelligence from Physical Simulators and World Models. arXiv preprint arXiv:2507.00917v3. https://arxiv.org/abs/2507.00917v3

[142] Luo, H., et al. (2026). Being-H0.5: Scaling Human-Centric Robot Learning for Cross-Embodiment Generalization. arXiv preprint arXiv:2601.12993v1. https://arxiv.org/abs/2601.12993v1

[143] Zheng, R., et al. (2026). EgoScale: Scaling Dexterous Manipulation with Diverse Egocentric Human Data. arXiv preprint arXiv:2602.16710v1. https://arxiv.org/abs/2602.16710v1

[144] Ding, K., et al. (2026). Agent Skill Evaluation and Evolution: Frameworks and Benchmarks. arXiv preprint arXiv:2606.11435v1. https://arxiv.org/abs/2606.11435v1

[145] He, F., et al. (2024). The Emerged Security and Privacy of LLM Agent: A Survey with Case Studies. arXiv preprint arXiv:2407.19354v2. https://arxiv.org/abs/2407.19354v2

[146] Chhabra, A., et al. (2025). Agentic AI Security: Threats, Defenses, Evaluation, and Open Challenges. arXiv preprint arXiv:2510.23883v3. https://arxiv.org/abs/2510.23883v3

[147] Xu, Y., et al. (2026). Toward Personalized LLM-Powered Agents: Foundations, Evaluation, and Future Directions. arXiv preprint arXiv:2602.22680v2. https://arxiv.org/abs/2602.22680v2

[148] Zhang, Y., et al. (2024). Vision-and-Language Navigation Today and Tomorrow: A Survey in the Era of Foundation Models. arXiv preprint arXiv:2407.07035v2. https://arxiv.org/abs/2407.07035v2

[149] Jiang, S., et al. (2025). A Survey on Vision-Language-Action Models for Autonomous Driving. arXiv preprint arXiv:2506.24044v1. https://arxiv.org/abs/2506.24044v1