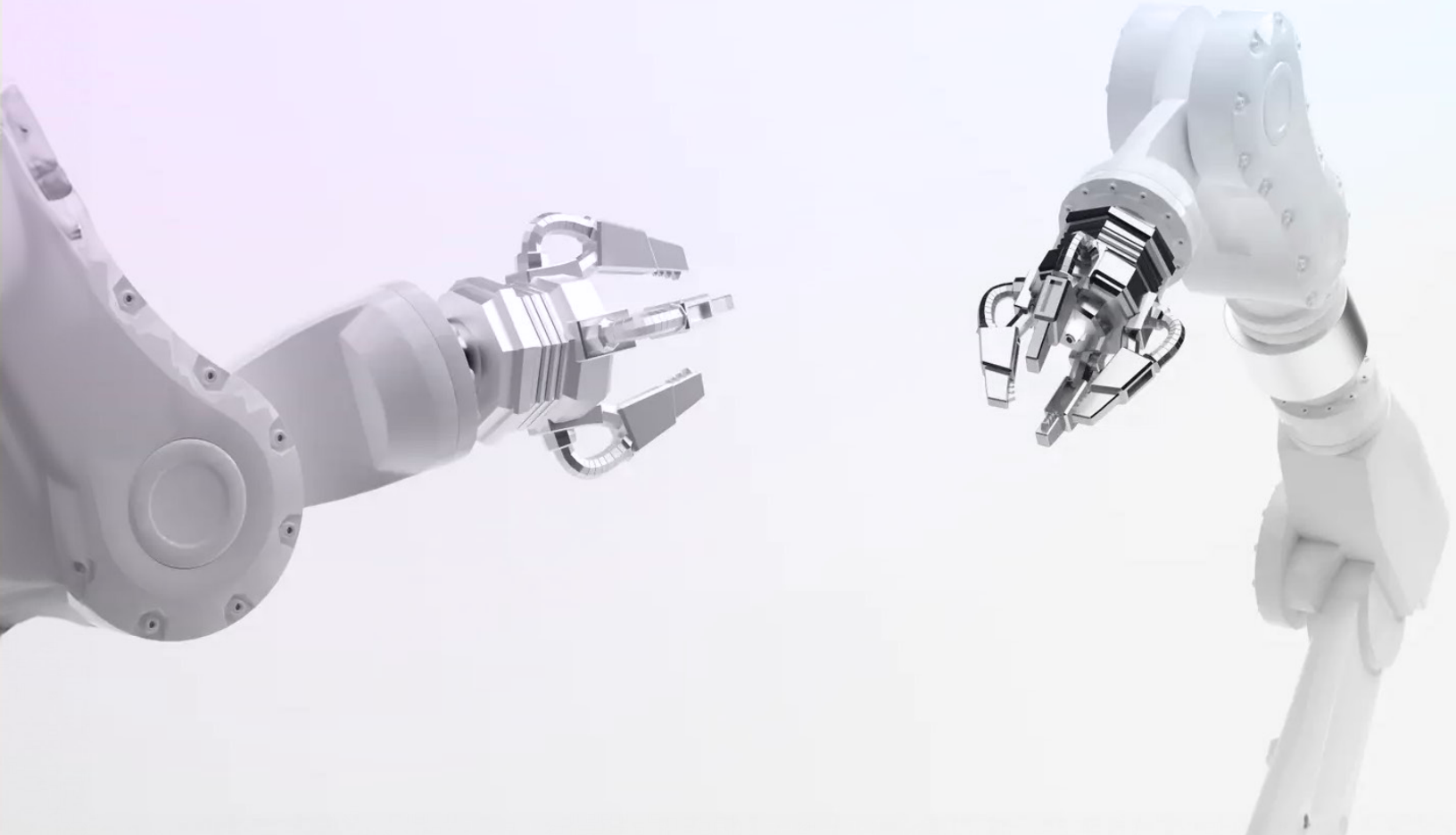
arm

White Paper
2026

The missing science of robotic systems

Capability integration in the model-centric era

Federico Pecora, July 21, 2026



The missing science of robotic systems

Capability integration in the model-centric era

Federico Pecora
July 21, 2026

Abstract

Robotics is entering a model-centric era: foundation models, vision-language-action policies, diffusion action generators, world models, and agents are expanding what robots can perceive, reason about, and do. But embodied intelligence remains a systems problem. A capable robot is a time-bounded assembly of perception, state estimation, reasoning, planning, control, safety, interaction, and orchestration, all constrained by latency, bandwidth, memory, power, reliability, and safety.

This review argues that robotics is becoming an integration-centric systems discipline. Scaling foundation models is a powerful source of capability, but it is not by itself an architecture for robots that must be reliably competent. Capability depends not only on models and algorithms, but on the state they consume, the timing bands they occupy, the compute and memory they use, the interfaces they expose, and the constraints under which they act. We organize the field around four questions: how capabilities are realized, composed in time, composed across compute and memory, and governed. Synthesizing work in robotics, embodied AI, runtimes, scheduling, model serving, middleware, and safety assurance, we identify open problems in semantic quality of service, preemptable inference, shared-state coherence, real-time generative models, governance compilation, and adaptive orchestration. Competent, general, and trustworthy robots require a science of capability integration for embodied AI.

Executive summary

Today, robotics is increasingly being shaped by foundation models, vision-language-action systems, generative policies, world models, and more capable on-device compute. These advances are expanding what robots can perceive, reason about, and do, while contemporary learned policies demonstrate useful closed-loop behavior on physical robots. What remains underdeveloped is the systems science needed to make perception, state estimation, learned policies, reasoning, planning, control, safety, interaction, and orchestration work together continuously under real-world constraints.

A robot needs to process sensor data, run AI models, update its understanding of the environment, plan actions, enforce constraints, and execute control loops concurrently. Each of these functions operates at different rates and places different demands on latency, power, memory bandwidth, compute resources, and reliability. Understanding these systems is a prerequisite for designing the right compute architecture for robot intelligence. The “robot brain”

is not simply the largest model that fits on an accelerator; it is the compute and memory architecture that can support the robot's capabilities together, at their required rates and levels of assurance.

This report examines these challenges in detail, looking at how robotic capabilities are realized, how they run at different speeds, how they are placed across heterogeneous compute and memory resources, and how they are governed at runtime. Specifically:

- **Capability realization:** Robotic capabilities can be delivered through learned models, planners, controllers, optimization methods, runtime assurance mechanisms, or a combination of these approaches. Each choice creates different requirements for latency, memory, bandwidth, compute, orchestration, and safety.
- **Timing and coordination:** Robots must coordinate workloads that operate at different speeds, from real-time control loops to perception, planning, reasoning, diffusion action generation, and world-model updates. These timing differences shape scheduling, pre-emption, buffering, state freshness, and system responsiveness.
- **Compute and placement:** Future robots will depend on heterogeneous compute across CPUs, GPUs, NPUs, microcontrollers, accelerators, memory systems, and real-time domains. Workload placement, data, movement, shared state, memory residency, scheduling, and isolation will be central to performance and reliability.
- **Safety and governance:** As more behavior is produced by learned models and agentic systems, robots will need runtime mechanisms that can enforce safety constraints, arbitrate between competing actions, and support inspectable, interruptible, and reliably constrained operation in real-world environments.

The main takeaway is that while model accuracy and task success remain important, real-world performance will depend on how capabilities are realized, composed, placed, and governed – and whether the underlying compute platform can support their inference, sensor processing, control, safety, recovery, and orchestration within strict limits for latency, power, memory bandwidth, and reliability.

As physical AI moves from model breakthroughs to real-world robotics, the industry will need a compute platform that can handle intelligence in motion. Robotics will increasingly depend on scalable, power-efficient heterogeneous compute that can support advanced AI alongside control, safety, and orchestration while meeting the practical constraints of physical systems.

1 Introduction

Robotics is entering an unusually exciting period. Advances in foundation models, vision-language-action policies, generative action models, world models, embedded accelerators, and heterogeneous compute platforms are changing what robots can perceive, infer, plan, and execute. Yet robotics differs from many other AI domains in one decisive respect: intelligence must be integrated into a physical, time-bounded, safety-constrained system. Contemporary learned policies already demonstrate useful closed-loop behavior on physical robots. What remains underdeveloped is the systems science for making perception, state estimation, learned policies, reasoning, planning, control, safety, interaction, and orchestration work together continuously and reliably under latency, power, memory-bandwidth, cost, and reliability constraints.

This systems understanding is a prerequisite for building the future compute architectures of physical AI. The “robot brain” is not simply the largest model that fits on an accelerator; it is the compute and memory architecture that can support the robot’s capabilities together, at their required rates and levels of assurance. Understanding how robot intelligence is realized, composed, and governed is therefore necessary to determine the right mix of CPUs, GPUs, NPUs, microcontrollers, memory, interconnect, isolation, and scheduling.

The central claim of this white paper is that robotics is becoming an integration-centric systems discipline. The realization of capabilities, their composition in time and across compute resources, and the governance of their behavior are all integration problems. A capability is not fully defined by the algorithm that implements it; it is also defined by the state it consumes, the timing band in which it operates, the compute resources on which it resides, the other capabilities with which it is coupled, and the constraints under which it may act.

This integration problem is becoming harder because the boundaries that once made robot architectures analyzable are beginning to blur. Vision-language-action models can colocate perception, language grounding, semantic reasoning, and action synthesis inside a learned policy, crossing lines that classical architectures treated as module boundaries. Agentic methods can dynamically construct, revise, and delegate pieces of the robot’s decision process, replacing fixed execution graphs with event-driven populations of reasoning tasks. At the same time, heterogeneous on-board compute makes the placement of workloads across CPUs, GPUs, NPUs, MCUs, memory hierarchies, and real-time domains a contextual decision rather than a static implementation detail.

These trends do not remove the need for integration; they make integration the core scientific problem. As capability boundaries become more fluid, the type, intensity, and timing of workloads become more dynamic. A robot may need to combine a real-time control loop, accelerator-resident perception, intermittent VLA inference, diffusion-based action proposals, a world-model

update, several short-lived agents, and safety monitors over shared state. This creates pressure not only on raw compute, but on coupling, concurrency, memory bandwidth, shared-state coherence, scheduling, preemption, and runtime orchestration.

Governance must evolve with this shift. Classical safety guards, formal specifications, and runtime assurance mechanisms remain essential, however they were developed for systems with clearer module boundaries, more predictable timing, and more explicit control authority. Future robots need governance mechanisms that can constrain learned policies, agentic planners, optimization-based controllers, human commands, and safety monitors operating at different rates and on different compute resources. The result is not a retreat from formal methods or safety engineering; it is a call to integrate them more deeply into the runtime architecture of embodied AI.

This white paper proposes a systems-oriented perspective on embodied AI, attempting to synthesize robotics, embodied AI, runtime systems, heterogeneous scheduling, model-serving systems, and safety assurance literature. Its organizing claim is that the path to competent, general, and trustworthy robots depends on developing a science of capability integration: a way to reason about how heterogeneous capabilities are realized, composed, placed, and governed as one embodied system.

The paper makes four contributions:

1. We frame robotics as an integration-centric systems discipline, where capability is shaped by coupling, timing, placement, shared state, and runtime constraints rather than by model performance alone.
2. We use four integration questions as a review structure for connecting robotics, embodied AI, runtime systems, heterogeneous compute, model serving, and safety assurance literature.
3. We synthesize contemporary evidence about compute provisioning in robotics around capability coupling, workload co-activation, memory bandwidth pressure, and runtime orchestration.
4. We identify open systems problems in semantic quality-of-service, preemptable inference, shared-state coherence, real-time generative models, governance compilation, and adaptive orchestration.

2 From layered autonomy to four questions for integration

Classical robot architectures already recognized the need to decompose intelligence across time scales and control layers. Reactive approaches such as subsumption emphasized fast, layered behaviors (Brooks 1986); hybrid architectures such as 3T separated deliberative planning, sequencing, and reactive control (Bonasso et al. 1997); behavior trees provided a modular representation for task-level orchestration and execution (Colledanchise and Ögren 2018). These ideas remain important, but the substrate has changed. Modern systems integrate large multimodal models, diffusion policies, learned world models, optimization-based control, real-time middleware, and heterogeneous accelerators.

The historical lesson is not that a single architecture won. It is that temporal hierarchy, explicit execution structure, and safety-aware arbitration reappear whenever robots must act in the physical world under latency, uncertainty, and resource constraints. Classical architectures largely assumed that designers would decide which capabilities existed, how they were sequenced, where they ran, and how they were supervised. Contemporary embodied systems make each of those choices less static. Capabilities may be realized by learned policies, optimization, search, or symbolic reasoning; temporal structure may be generated or revised by agents; spatial placement may shift across CPUs, accelerators, memory hierarchies, and real-time islands; and governance must remain enforceable even when behavior is proposed by opaque or stochastic models.

We argue that advancing the science of integration should become an important research goal for academia and industry in the coming years: understanding how robot capabilities are realized, composed, distributed, and governed across heterogeneous compute systems. This goal is central to designing effective compute architectures for robotics, because capability choices determine accelerator demand, resident memory, bandwidth pressure, CPU orchestration load, timing constraints, and safety-critical runtime interactions.

We break this goal into four questions, each addressing a different part of the compute-requirement space:

- **How are capabilities realized?** Progress comes not only from new models, but from how learning, control, optimization, planning, and runtime assurance are combined. The realization choice determines whether pressure appears primarily as accelerator work, resident model memory, bandwidth, CPU orchestration, or real-time control demand, from classical hybrid architectures (Bonasso et al. 1997; Colledanchise and Ögren 2018) to contemporary systems that combine language, vision, action generation, and affordance grounding (Ahn et al. 2022; Driess et al. 2023; Brohan et al. 2023).

- **How are capabilities composed in time and function?** Robot capabilities operate across a range of latency and frequency bands, and their cadences may become contextual when multiple realizations are available. The timing relationships among real-time control and MPC (Mayne 2014), perception pipelines, language-mediated planning, diffusion action generation, and runtime monitors determine concurrency, preemption, buffering, stale-state risk, and scheduling pressure.
- **How are capabilities composed across systems and compute?** Effective robots require multiple compute types and multiple memory domains, from real-time CPUs and embedded controllers to accelerator-resident perception and model serving (Dao et al. 2022; Kwon et al. 2023). When capability realization is contextual, placement also becomes dynamic: the runtime must decide what remains ring-fenced, what can be shared, what must be copied, and where bandwidth or orchestration bottlenecks appear.
- **How are capabilities governed and constrained?** Some behavioral requirements are best learned from data, especially preferences that are hard to specify directly. Others are accurately specifiable, safety-critical, or introduced online, and should be expressible as runtime constraints or assurance mechanisms (Ames et al. 2019; Seto et al. 1998; Bak et al. 2014; Alshiekh et al. 2018). These requirements must remain enforceable across dynamic capability hierarchies, while the compute cost of enforcement, from differentiable specifications to formal verification, becomes part of the systems problem.

Together, these questions imply that system-level understanding requires building, tracing, measuring, and validating integrated systems rather than evaluating isolated components alone (Casini et al. 2019; Bédard et al. 2022; Bédard and Dagenais 2023; Macenski et al. 2023). The next four sections address the questions in turn. Each section reviews representative technical directions, then examines what plausible answers imply for accelerator work, resident memory, bandwidth, CPU orchestration, timing, and governance pressure.

For the provisioning discussion, we use three reference levels of deliberation:

- **Advisory deliberation:** a sparse reasoning layer that interprets scenes, instructions, task failures, or recovery options, but does not continuously sit in the sense–reason–act loop.
- **Interactive deliberation:** a reasoning layer queried frequently enough to shape task execution, combine semantic reasoning with action proposals, and participate in user interaction or recovery.
- **Continuous deliberation:** a high-duty-cycle layer that keeps deliberative models, action generation, world modeling, and shared semantic state active.

These are not proposed product categories; they are reference points for asking how resource pressure changes as deliberation moves from occasional advice to interactive participation and then to continuous operation.

The rest of the paper uses these questions to argue that integration pressure is not a secondary implementation concern, but a primary determinant of what robot capabilities can be realized in practice.

3 How are capabilities realized?

This question asks which paradigms should realize a capability for a given robot, task, and environment. The integration issue is that each choice creates different pressure on latency, memory, bandwidth, orchestration, and governance.

Recent embodied AI work demonstrates the breadth of realization paradigms: vision-language-action models and multimodal foundation models (Brohan et al. 2023; Kim et al. 2024; Driess et al. 2023); language-grounded planning and affordance reasoning (Ahn et al. 2022); diffusion-based visuomotor policies (Chi et al. 2025); latent world models and transformer world models (Hafner et al. 2023; Micheli et al. 2023); and continual embodied agents that acquire skills over long horizons (Wang et al. 2023).

No single paradigm dominates across the robotics stack. High-frequency control demands deterministic execution and numerical stability, as reflected in real-time control and MPC formulations (Mayne 2014). Perception and semantic mapping are often accelerator-intensive and bandwidth-sensitive, as shown by tracing and zero-copy acceleration work in robotics pipelines (Lajoie et al. 2022; NVIDIA 2026a, 2026b). Task-level reasoning may be symbolic, learned, language-driven, or hybrid (Bonasso et al. 1997; Ahn et al. 2022; Liang et al. 2023). Motion planning increasingly combines search, optimization, and learned proposals (LaValle 2006; Carvalho et al. 2025; Chi et al. 2025). Safety and governance require enforceable runtime mechanisms, such as barrier functions (Ames et al. 2019), runtime assurance (Bak et al. 2014), Simplex switching (Seto et al. 1998), and shields (Alshiekh et al. 2018).

One important branch of recent work attempts to absorb more of the planning-to-action pipeline into a single large learned policy, typically a vision-language-action model. RT-2 (Brohan et al. 2023), OpenVLA (Kim et al. 2024), RT-X (Open X-Embodiment Collaboration 2023), π_0 (Black et al. 2025), GR00T N1 (Björck et al. 2025), and Xiaomi-Robotics-1 (Guo et al. 2026) all represent variants of this model-maximalist trajectory: perception, language grounding, semantic reasoning, and action synthesis are increasingly colocated inside a generalist policy trained across broad task and embodiment distributions. This approach is important for the integration thesis because it partially changes what must be composed explicitly. A strong VLA can provide

competent baseline behavior with a simplified external interface, reducing the amount of hand-written task logic and skill wiring required for many common tasks.

The caveat is that internalizing composition inside a learned model does not eliminate integration; it moves the boundary. Distributional breadth is not the same as behavioral adequacy for a given deployment. A VLA may cover many objects, instructions, and embodiments, while still being insufficiently precise, auditable, interruptible, or robust under local lighting, contact, timing, safety, or human-interaction conditions. The systems question therefore becomes how a model-maximalist policy is wrapped: what runtime monitors observe it, what constraints bound it, what controllers or policies refine its actions, when fallback controllers override it, what state it reads and writes, and how its confidence, latency, and failure modes are exposed to the rest of the robot.

Xiaomi-Robotics-1 illustrates both the progress enabled by scaling and the remaining competence gap (Guo et al. 2026). The system uses more than 100,000 hours of embodiment-free manipulation trajectories together with cross-embodiment post-training data. In a controlled model-scaling experiment using 20,000 pre-training hours for each variant, its out-of-the-box real-robot success increased from 61% for a 2B-parameter model to 79% for a 10B-parameter model on four task categories seen during post-training but evaluated with unseen environments and object instances. For four tasks held out from the in-house robot dataset, downstream fine-tuning on 36 demonstration-hours in total—less than 10 hours per task on average—produced a 75% average success rate over 10 trials per task, compared with 40% for the reported $\pi_{0.5}$ baseline. This is a substantial improvement in data efficiency, but it is not zero-shot acquisition of new skills or deployment-level reliability: one in four evaluated trials remained unsuccessful on average, and adaptation still required task-specific demonstrations and fine-tuning. The scope of the result is also limited by the evaluation protocol: the paper does not report deliberate lighting variation or the number and type of physical sites; the held-out-task study uses only 10 trials per task; and its milestone-based success criteria are not accompanied by acceptance tolerances, cycle-time distributions, latency, energy, throughput, intervention, or recovery measures. The reported results therefore establish scaling and data-efficient adaptation under this protocol, but do not establish industrial readiness or economic viability.

Agentic AI already provides a useful analogy. We do not expect an agent to reimplement a filesystem operation neurally when the operating system provides a reliable primitive; the agent should call the primitive. At the same time, we do expect agents to invoke learned generative models when the task is intrinsically perceptual or semantic, such as image editing or multimodal interpretation. Robotics should be treated the same way. Generalist policies, VLAs, diffusion models, planners, controllers, monitors, and verified runtime mechanisms are all candidate primitives for embodied agents. A model-maximalist architecture may simplify some interfaces, but forcing every capability through a single learned policy would discard mature structure where

it is useful and slow progress where explicit mechanisms, tools, or controllers are already the better abstraction.

Choosing and wrapping the right primitives becomes harder as capability structure is increasingly constructed online. Agentic methods can assemble monitors, recovery procedures, tool calls, plans, and interaction loops dynamically rather than relying only on boundaries designed into the system ahead of time. The relevant comparison is therefore not only between static architectural paradigms, but also between systems whose capability structure is fixed at design time and systems whose structure is partly produced at runtime.

Recent embodied runtime work makes this boundary shift more concrete. Embodied.cpp provides an emerging portable inference substrate for embodied models, including common model packaging, typed input adapters, action chunks, split-inference hooks, persistent model state, and phase-level latency telemetry (Xu et al. 2026; SEU-PAISys 2026). These mechanisms are useful evidence that diverse VLA and world-action-model families can share part of an execution contract. The current open-source implementation nevertheless remains primarily a synchronous model-serving layer: multi-rate orchestration, adaptive placement, robot-facing deployment contracts, and runtime governance remain architectural goals rather than implemented runtime services. Although not a complete solution to capability integration, Embodied.cpp provides evidence for the feasibility of a common embodied inference substrate.

3.1 Provisioning consequences

We can make the numerical pressure created by different realization choices more explicit by tracking four resource classes. Accelerator work captures the burst tensor-compute demand placed on GPUs, NPUs, or other matrix engines for one active inference episode, proposal, or update. Resident memory captures the capacity needed to keep model weights, caches, sensor tensors, latent state, and runtime buffers available without continual reload. Memory bandwidth captures the active-phase movement of those weights, activations, sensor streams, caches, and shared state through the memory system. Application CPU captures orchestration, middleware, tool use, planning glue, serialization, scheduling, monitoring, and the control-plane side of agentic workloads, including spawning agents, issuing tool calls, querying state, maintaining short-lived context, and arbitrating among proposals. Let the incremental deliberation requirement for resource $k \in \{\mathrm{accel}, \mathrm{mem}, \mathrm{BW}, \mathrm{CPU}\}$ be

$$R^{k}_{\mathrm{delib}} = R^{k}_{\mathrm{AR}} + R^{k}_{\mathrm{diff}} + R^{k}_{\mathrm{world}} + R^{k}_{\mathrm{int}} \qquad (1),$$

where the terms represent autoregressive reasoning, diffusion/action generation, world-model prediction, and integration overhead. For $k = \mathrm{CPU}$, this decomposition is supplemented by an additional orchestration/control-plane term $R^{\mathrm{CPU}}_{\mathrm{orch}}$, which captures prompt construction, tool invocation, state access, serialization, scheduling, monitoring, and model-serving work. This CPU term grows roughly with the number of active agents or reasoning episodes before contention,

shared queues, locks, allocators, middleware executors, and model-serving gateways can introduce nonlinear latency.

The component terms should be read as resource pressures, not as a calibrated hardware model. R^k_{AR} covers autoregressive VLM/VLA inference, including accelerator work, resident weights and caches, and active memory traffic. R^k_{diff} covers diffusion/action-generation proposals, R^k_{world} covers world-model prediction and retained latent state, and R^k_{int} covers integration overhead from serving boundaries, synchronization, copies, and shared-state access. We report here and in the following sections the resulting reference envelopes but omit the intermediate accounting: the estimates depend on assumptions about token budgets, KV-cache residency, active fractions, diffusion denoising steps, proposal rates, world-model rollout branching, and memory-traffic multipliers. The Appendix gives the explicit back-of-the-envelope calculations and introduces those auxiliary quantities.

For autoregressive VLM/VLA realization, the dominant pressures are active parameter count, token budget, context length, KV cache, multimodal inputs, and duty factor. A compact advisory model already requires TFLOP-scale per active episode, while a 7B-class interactive model moves into the several-TFLOP regime before invocation rate or overlap is considered. With 8-bit weights, weights and KV cache for a 7B-class model are already on the order of 10 GB before vision buffers and runtime scratch space are considered. Active-phase bandwidth can be more constraining still: for a 7B-class batch-1 decode path, repeatedly streaming weights at interactive token rates places the requirement in the few-hundred-GB/s range before KV, activation, prompt, vision, and burst factors are included (Kwon et al. 2023; Dettmers et al. 2022; Frantar et al. 2023; Chen 2026). The calculations behind these statements are given in the Appendix.

Diffusion and world-model realizations create different pressures. Diffusion action generation repeats denoising over state-conditioned action sequences (Chi et al. 2025; Janner et al. 2022); a compact proposal is a few-TFLOP episode, while repeated state and feature movement can add roughly 100 GB/s of pressure at interactive proposal rates. World models add rollout length, branch count, retained latent state, temporal history, and runtime buffers (Hafner et al. 2023; Micheli et al. 2023; Zidan et al. 2026). A DreamerV3-style scale suggests resident state in the several-GB regime and active rollout/update work from a few TFLOP to the low tens of TFLOP for modest branching.

Finally, we represent integration overhead as a separate sensitivity term:

$$R^k_{\mathrm{int}} \approx \eta^k_{\mathrm{int}}\left(R^k_{\mathrm{AR}} + R^k_{\mathrm{diff}} + R^k_{\mathrm{world}}\right) \qquad (2),$$

with η^k_{int} interpreted as the additional workload induced by model serving, adaptation layers, message transport, copying, serialization, monitoring, synchronization, and shared-state access. It does not represent the gap between achieved workload performance and advertised

hardware peak. Robotics middleware studies show that node composition, tracing, message flow, and IPC strategy materially affect latency and resource use (Macenski et al. 2023; Bédard et al. 2022; Bédard and Dagenais 2023; Kronauer et al. 2021), and recent zero-copy frameworks explicitly target this overhead (NVIDIA 2026b; Ishikawa and Kato 2025). These studies motivate representing integration overhead explicitly, but they do not determine universal coefficients for η_{int}^{k}. In the rounded reference envelopes below, we set $\eta_{\text{int}}^{\text{accel}} = 15\%$, $\eta_{\text{int}}^{\text{mem}} = 30\%$, and $\eta_{\text{int}}^{\text{BW}} = 30\%$ as illustrative upper assumptions; the Appendix gives the sensitivity ranges used in the underlying arithmetic.

Table 1 summarizes the component-level anchors introduced in this section. It deliberately does not yet assign them to sparse, interactive, or continuous autonomy levels. The point is to keep the realization question separate from the temporal-composition question: AR reasoning, diffusion action generation, world modeling, and integration overhead each introduce different resource terms before the runtime decides how often they run and how much they overlap.

Table 1. Rounded component-level provisioning anchors for the deliberation terms in Equation 1. Accelerator values are approximate FP8-equivalent burst work per active episode, proposal, or rollout bundle. Memory values are resident capacity. Bandwidth values are active-phase pressure where a rate assumption is needed. Exact calculation ranges are listed in the Appendix.

Component	**Accelerator anchor**	**Memory anchor**	**Bandwidth anchor**	**CPU anchor**
Autoregressive VLM/VLA	Around 1 TFLOP for compact advisory use; several TFLOP for 7B-class reasoning	Several GB for compact advisory use; order 20 GB for 7B-class use with KV and runtime buffers	Few hundred GB/s for interactive use; up to roughly half a TB/s for richer near-continuous use	Prompt assembly, context packaging, serving dispatch, tool-mediated control flow
Diffusion action generator	Few TFLOP per compact action proposal	Sub-GB to a few GB depending on denoiser size, visual encoder, features, and scratch space	Order 100 GB/s at interactive proposal rates with repeated state access	Proposal scheduling, receding-horizon handoff, trajectory-buffer management
World model	Few to low-tens TFLOP per rollout/update bundle	Sub-GB to several GB for weights, latent state, branches, history, and runtime buffers	Tens to low hundreds of GB/s with latent/history traffic	Rollout management, branch selection, state-history bookkeeping
Integration overhead	$R^{accel}_{int} \leq 5$ TFLOP ($\eta = 15\%$)	$R^{mem}_{int} \leq 10$ GB ($\eta = 30\%$)	$R^{BW}_{int} \leq 300$ GB/s ($\eta = 30\%$)	Message transport, copying, synchronization, monitoring, shared-state access

These component anchors expose a design choice. A powerful VLA, a diffusion action generator, and a continuously updated world model can all be justified by task performance, but they compete for the same accelerator residency, memory bandwidth, and scheduling slack. The question is therefore not simply whether a larger VLA is better, but whether the marginal capability gained by enlarging the VLA is worth the capacity it removes from world-model prediction, safety monitoring, fallback evaluation, or other integration functions.

Accelerator work grows quickly in the examples, yet it is also the resource most directly addressed by choosing a larger accelerator or a lower-precision inference path. Resident memory and active-phase bandwidth requirements are harder to compensate for: they determine which models, caches, sensor tensors, world-model state, and safety monitors can coexist without repeated reloads or copies. CPU pressure is less cleanly quantified, but it is dangerous because it sits on the scheduling and orchestration path; overloaded control-plane work can turn into jitter, stale state, missed callbacks, or delayed arbitration. The practical headroom is therefore often less about whether one more model can be executed in isolation, and more about whether model scale, world modeling, runtime assurance, and agentic orchestration can be co-resident and co-active under latency, power, and thermal limits. As deliberation moves from advisory to continuous use, bandwidth, residency, and orchestration grow fast enough to force architectural tradeoffs among model scale, world modeling, safety, and runtime assurance.

3.2 Design implications and open questions

Capability realization is no longer only an algorithm-selection problem. A VLA, diffusion policy, world model, optimizer, symbolic planner, or controller may all be reasonable ways to realize a capability, but each choice creates a different downstream systems burden. The component estimates above show why: a 7B-class AR component can require about 10 GB for weights and KV cache before vision buffers, while active-phase bandwidth for interactive VLM/VLA use can already sit in the hundreds of GB/s. The central takeaway is that capability quality must be evaluated together with resident memory, bandwidth, latency, CPU orchestration, and the cost of wrapping the capability with state estimation, safety, recovery, and governance.

Three broad questions follow:

- When should a capability be internalized inside a generalist learned policy, and when should it remain an explicit planner, controller, monitor, or recovery mechanism?
- What runtime interface should learned capabilities expose so the rest of the robot can budget their latency, uncertainty, state dependencies, action envelopes, and failure modes?

- How should robotics benchmarks report systems pressure alongside task success, so capability gains can be compared against memory, bandwidth, timing, and orchestration costs?

4 How are capabilities composed in time and function?

This question asks how capabilities should be organized across timing bands and functional layers. The integration issue is that control, perception, reasoning, action generation, and governance run at different duty cycles.

The core systems issue is that robot capabilities operate at different duty cycles and decision cadences. Torque loops may run at hundreds or thousands of hertz, model-predictive control and trajectory optimization at tens of hertz, perception pipelines at frame rates, semantic reasoning at interactive or intermittent cadences, and long-horizon planning at sparse cadences. This frequency mismatch makes temporal hierarchy unavoidable.

Embodied inference runtimes are beginning to expose this mismatch directly. The Embodied.cpp report (Xu et al. 2026) frames embodied deployment as a different runtime contract from conventional LLM or VLM serving: multi-rate execution inside closed-loop control, latency-first batch-1 inference on heterogeneous edge hardware, and extensible embodied interfaces rather than fixed token I/O. This framing supports the claim that timing is not an implementation detail added after model design, even though the current implementation does not yet provide a general multi-rate scheduler (SEU-PAISys 2026). Timing is part of what the capability is, because the same backbone, perception encoder, predictive branch, or action head may need different refresh policies and jitter bounds depending on its role in the robot.

Classical temporal composition assumes that a designer specifies the execution structure: a state machine, behavior tree, task planner, executive, or layered autonomy architecture determines which capability runs when. Agentic methods introduce a different regime. Here, language-model agents that interleave reasoning and acting (Yao, Zhao, et al. 2023), search-style deliberative planners (Yao, Yu, et al. 2023), reflective memory systems (Park et al. 2023), and multi-agent conversations (Wu et al. 2023) can dynamically construct, revise, and delegate pieces of the robot's decision process. In robotics, this appears as LLM-mediated task planning (Singh et al. 2023; Song et al. 2023), code-policy generation (Liang et al. 2023), inner-monologue reasoning (Huang, Xia, et al. 2023), value-map construction (Huang, Wang, et al. 2023), and large-scale orchestration of robot agents (Ahn et al. 2024).

This changes the compute equation in two ways. First, temporal composition becomes event-driven and opportunistic: agents may be spawned to monitor perception streams, coordinate non-real-time recovery after controller safety events, arbitrate degraded-mode behavior, interact with users, or revise task plans. Second, the dominant resource pressure is not always

dense accelerator work. Agentic loops create control-plane load: token-sequential inference, prompt construction, retrieval, state serialization, tool calls, sandboxed code execution, message routing, and many small concurrent decisions. Even when the underlying LLM or VLM uses an accelerator, the surrounding agent runtime can become CPU-heavy and latency-sensitive because it coordinates many short-lived reasoning tasks over shared robot state.

4.1 Provisioning consequences

The component anchors above become system requirements only after they are placed on a temporal schedule. A useful duty-cycle hierarchy spans sparse reasoning at 1–10 s, intermittent planning at 200 ms–2 s, interactive planning at 50–200 ms, continuous deliberation at 10–50 ms, and servo-level control below 10 ms. Moving a capability into a faster timing band changes the provisioning problem: there is less slack for batching, queueing, retries, or fallback, and the component must be active, resident, and synchronized with shared state more often. Designers may respond by reducing model size, context length, token budget, precision, or query frequency. Overall, tighter temporal integration forces capability quality, latency, residency, and bandwidth to be considered together. In the accounting detailed in the Appendix, temporal composition converts per-episode accelerator work into a sustained work rate and changes how much of the time bandwidth must be provisioned as active.

Table 2 composes the component anchors in Table 1 into three temporal reference design points. To make the comparison crisp, it selects the upper cadence, active fraction, branch count, component anchor, and integration coefficient from the working ranges in Appendix Table 5; the appendix retains the ranges for reference. For accelerator work, the per-episode, per-proposal, and per-rollout-bundle terms are multiplied by their invocation or update rates to obtain sustained FP8-equivalent TFLOP/s. For every resource, the component terms are additive when they are co-active.

Each row is grounded in public model sizes, configurations, or systems studies where possible. These are provisioning figures for the deliberative subsystem under the architectural assumption that baseline sensor processing, low-level perception, state estimation, control, and safety supervision are provisioned separately on dedicated hardware such as ISPs, DSPs, MCUs, and real-time controllers. Their compute and local memory are therefore not added to Table 2, although the cost of exchanging state with them remains part of integration overhead. ROS 2 tracing, message-flow, and perception-pipeline tracing studies show why this data movement and background execution still need to be measured (Bédard et al. 2022; Bédard and Dagenais 2023; Lajoie et al. 2022). The AR row combines transformer token scaling, 3B–7B-class VLM/VLA scales, 8-bit weights, KV-cache accounting, and active-phase traffic assumptions (Kaplan et al. 2020; Kwon et al. 2023; Dettmers et al. 2022; Frantar et al. 2023; Beyer et al. 2024; Bai et al. 2025; Kim et al. 2024). The diffusion row combines denoising-step counts, per-proposal work, Diffusion Policy action-generation structure, public image-configuration scales, and ResNet-18 encoder

footprint anchors (Chi et al. 2025; Real-Stanford Diffusion Policy Contributors 2023; PyTorch Contributors 2026). The world-model row uses DreamerV3-style model-size families, latent prediction, and imagined rollouts as anchors (Hafner et al. 2023; Micheli et al. 2023; Hafner 2025). The R^k_{int} row reports the resource demand induced by integration overhead; the parenthetical η^k_{int} gives the sensitivity factor used to estimate that demand. It represents overhead internal to the deliberative layer, such as model-serving boundaries, adaptation layers, copies, synchronization, and shared-state access. Composition, IPC, and zero-copy studies motivate treating this overhead explicitly (Macenski et al. 2023; Kronauer et al. 2021; NVIDIA 2026b; Ishikawa and Kato 2025).

Table 2. Conservative provisioning design points for the deliberative subsystem, assuming baseline sensing, low-level perception, state estimation, control, and safety supervision run on dedicated hardware. Values select the upper working assumptions in Appendix Table 5 and are rounded for readability. Accelerator entries are sustained FP8-equivalent work rates, not hardware nameplate ratings.

Resource	**Component**	**Advisory deliberation**	**Interactive deliberation**	**Continuous deliberation**
Assumptions	AR model size	3B parameters	7B parameters	7B parameters
	Diffusion model	—	4 GB resident	4 GB resident
	World-model size	—	—	400M parameters
	AR invocation rate	1 Hz	5 Hz	20 Hz
	AR active fraction	0.25	0.7	1.0
	Diffusion proposal rate	—	10 Hz	20 Hz
	World update/branching	—	—	5 Hz, 3 branches
Accelerator, FP8-equiv. TFLOP/s	R_{AR}	1	40	170
	R_{diff}	—	70	140
	R_{world}	—	—	90
	R^k_{int} (η = 15%)	0.2	20	60
	Total	**2**	**130**	**460**
Memory (GB)	R_{AR}	12	20	30
	R_{diff}	—	4	4
	R_{world}	—	—	8
	R^k_{int} (η = 30%)	4	7	13
	Total	**16**	**31**	**55**
Bandwidth (GB/s)	R_{AR}	100	300	600

	R_{diff}	—	100	200
	R_{world}	—	—	150
	R^{k}_{int} (η = 30%)	20	100	300
	Total	**100**	**500**	**1300**
CPU	Threads/processes	Episodic request handlers and serving dispatch.	Concurrent agent, tool, and serving processes.	Persistent queues, monitors, and arbitration processes.
	Control-plane work	Prompt assembly and sensor/context packaging.	Retrieval, serialization, routing, and callback scheduling.	Shared-state access, monitor coordination, and contention-induced jitter.

Within each column, the total is obtained by adding the AR, diffusion, world-model, and integration rows. Moving from advisory to interactive deliberation changes the active set and cadence by introducing frequent AR/VLA inference and diffusion-style action proposals; continuous deliberation further includes persistent world-model state and more continuous shared-state traffic. Thus the continuous accelerator design point is obtained by adding 170 TFLOP/s for AR, 140 TFLOP/s for diffusion, 90 TFLOP/s for the world model, and roughly 60 TFLOP/s for integration overhead, giving about 460 FP8-equivalent TFLOP/s. CPU orchestration pressure is harder to quantify, but it grows with agent fan-out, request rate, tool calls, serialization, scheduling, monitoring, model-serving calls, and access to shared state.

The design points assume that the deliberative model set becomes more sophisticated as its invocation rate and active fraction increase. Advisory deliberation uses a 3B-class AR advisor intermittently. Interactive deliberation uses a 7B-class VLM/VLA and an image-conditioned diffusion policy with a 4 GB resident-memory envelope, proposing actions at 10 Hz. Continuous deliberation retains the same AR and diffusion model classes and resident-memory assumptions rather than assuming larger versions, but keeps the VLA active at a higher cadence, doubles the diffusion proposal rate, and adds a 400M-parameter world model updating at 5 Hz with three rollout branches. Under these assumptions, the AR memory design point rises from 12 to 20 to 30 GB and its sustained bandwidth rises from roughly 100 to 300 to 600 GB/s across the three levels. These are provisioning points for increasing residency and traffic; they do not include a separately quantified increase in context length or sensory input richness. Diffusion, world-model, and integration requirements are then added as shown in Table 2.

Diffusion and world models make the same point in different temporal regimes. An interactive diffusion action generator at around 10 Hz with a handful of denoising steps executes many denoising passes per second before concurrency factors are applied. This rate is meant to

represent repeated short-horizon action proposals at an interactive planning cadence. Using the diffusion bandwidth accounting detailed in the Appendix, repeated state and conditioning access can add order-100-GB/s pressure even before model weights and AR inference are considered. In the continuous deliberation reference level, a deliberative world-model rollout/update process adds a persistent stateful component: few-Hz branched updates create multiple latent rollout/update paths per second plus history and branch-buffer traffic. This lower rate should not be read as intrinsic to world models: if the world model participates in faster receding-horizon planning, safety prediction, or action selection, the same terms scale upward with update frequency and branching. The provisioning implication is large: temporal composition can move the bandwidth requirement from a tens-of-GB/s advisory load to several hundred GB/s once repeated AR inference, diffusion proposals, and then world-model state traffic become co-active.

Agentic composition adds a third multiplier that is mostly CPU and orchestration pressure. If one agent monitors perception, another coordinates recovery, another handles user dialogue, and another revises the task plan, the AR accelerator may still be shared, but prompt construction, token streaming, multimodal preprocessing, model-serving calls, tool invocation, retrieval, serialization, queueing, and access to shared state replicate across requests and agents. The implication is stronger than "LLMs add compute": temporal composition becomes an admission-control problem over a mixed workload of hard real-time loops, soft real-time perception, accelerator-bound inference, and CPU-heavy control-plane tasks.

Qualitatively, Table 2 places the advisory design point at approximately 2 dense FP8-equivalent TFLOP/s, 16 GB, and 100 GB/s; the interactive point at 130 TFLOP/s, 31 GB, and 500 GB/s; and the continuous point at 460 TFLOP/s, 55 GB, and 1.3 TB/s. Advertised dense FP8 tensor throughput, memory capacity available to the deliberative subsystem, and peak memory bandwidth must each be at least the corresponding value. Even then, the table values are optimistic lower bounds on hardware nameplate requirements: real batch-1 multimodal workloads do not sustain ideal tensor occupancy or peak streaming bandwidth; installed memory is not wholly available to model state; and implementations mix FP8 with BF16, FP16, integer, and non-tensor operations. Dense rather than sparsity-enhanced ratings must be used.

Published optimized Transformer baselines give a scale for this gap. Large-batch prefill has reached 76% model-FLOP utilization, while a highly optimized paged-attention implementation reports up to 73% model-FLOP utilization in prefill and 86% memory-bandwidth utilization in decode (Pope et al. 2022; Jiang et al. 2026). Taking the less favorable reported accelerator utilization and the reported bandwidth utilization gives reference multipliers of approximately 1.4 for accelerator throughput and 1.16 for bandwidth. These measurements come from optimized single-workload Transformer benchmarks, each dominated by one inference phase, rather than concurrent execution of heterogeneous models and runtime functions. The multipliers should

therefore be read as lower bounds on nameplate inflation for robotics, not as expected utilization factors. Applying them to Table 2 gives approximately 2.8, 182, and 644 advertised dense FP8 TFLOP/s for the three design points, and approximately 116, 580, and 1,508 GB/s of peak bandwidth. Mixed, batch-1, deadline-constrained robotic workloads will require larger multipliers, although the cited results do not quantify how much larger. Even at these lower bounds, continuous deliberation is already in the 0.6-PFLOP/s accelerator and 1.5-TB/s bandwidth class; additional robotics headroom or model concurrency can push accelerator requirements into PFLOP/s territory.

Utilization measurements do not establish a useful memory-capacity multiplier. PagedAttention demonstrates that specialized allocation can make KV-cache waste nearly zero (Kwon et al. 2023), but a robot runtime must still accommodate allocator fragmentation, runtime reservations, and transient buffers. We can therefore posit that at least 16, 31, and 55 GB must be available to the deliberative subsystem, and installed capacity must be higher by an amount measured for the target runtime.

4.2 Design implications and open questions

Temporal composition determines whether a deliberative capability is occasional advice, repeated participation, or a continuously active part of the robot. The same model can be manageable when advisory and difficult to provision when it becomes interactive or continuous: the reference envelopes move from around 100 GB/s for advisory deliberation to several hundred GB/s for interactive deliberation and can approach or exceed 1 TB/s for continuous deliberation. The central takeaway is that robot runtimes need to treat timing, duty cycle, preemption, and degradation as first-class properties of inference and reasoning, not only of controllers.

Three broad questions follow:

- How should hard real-time control, soft real-time perception, generative reasoning, diffusion proposals, and world-model updates be scheduled under shared resource constraints?
- How should agentic reasoning tasks be admitted, prioritized, bounded, and arbitrated when many short-lived agents compete for CPU control-plane work and shared state?
- How can inference and reasoning be made interruptible, preemptable, resumable, or degradable while preserving enough semantic value for safe embodied action?

5 How are capabilities composed across systems and compute?

This question asks where capabilities execute and where their state should reside. The integration issue is deciding what to share, isolate, or move as capabilities are realized and dynamically composed.

Robot capabilities are inherently distributed: sensing, control, planning, and reasoning may execute across CPUs, GPUs, NPUs, MCUs, real-time islands, memory hierarchies, and interconnect fabrics within the same embodied system.

Earlier cloud and fog robotics work framed spatial distribution primarily across robots, infrastructure, and remote compute resources (Kehoe et al. 2015; Hu et al. 2012). The central pressure in contemporary embodied systems is increasingly internal to the robot: heterogeneous on-board compute, CPU/GPU/NPU/RT partitioning, runtime scheduling, orchestration, shared-state execution, dynamic inference, workload placement, memory systems, model serving, and accelerator scheduling. In this sense, spatial composition turns robotics into a runtime systems problem. The relevant concerns resemble operating systems and heterogeneous runtimes: scheduling, placement, shared memory, accelerator arbitration, model co-residency, cache/KV-cache management, data movement, and isolation. Physical robotics adds a further burden: optimization, reasoning, and inference are not one-off queries but recurring workloads that remain active around sensing, state estimation, planning, control, recovery, and safety. Even advisory deliberation sits on top of a continuously operating stack, and richer autonomy layers add varied compute-heavy workloads such as autoregressive inference, diffusion action generation, optimization, and world-state maintenance.

The strongest direct evidence in robotics currently appears at the middleware and runtime layer. ROS 2 node composition shows that placing nodes into shared processes can substantially change CPU, memory, and latency behavior while preserving modular software boundaries (Macenski et al. 2023). Executor and response-time analyses show that callback scheduling and processing-chain interference are not incidental implementation details, but central to whether robotic computation is analyzable and schedulable (Casini et al. 2019). ROS 2 tracing and message-flow analyses further show that distributed systems need instrumentation to reconstruct causal paths, executor state, delays, bottlenecks, and critical paths (Bédard et al. 2022; Bédard and Dagenais 2023). Tracing work in perception and mapping pipelines makes the same point at the application level (Lajoie et al. 2022). Industrial acceleration stacks make a similar point from the hardware side: Isaac ROS and NITROS use type adaptation, type negotiation, and zero-copy transport to reduce CPU copies and keep data resident in accelerator-friendly memory across ROS graphs (NVIDIA 2026a, 2026b). Static task-graph scheduling, real-time accelerator scheduling, and mixed robotics compute architectures all point

toward placement as a first-class design problem (Topcuoglu et al. 2002; Zou et al. 2025; Fil et al. 2026).

This evidence is important, but it is still mostly evidence about the substrate rather than a mature literature on spatial capability composition itself. The robotics literature has increasingly studied node composition, executor scheduling, tracing, middleware latency, and accelerator transport. It has less often studied the end-to-end placement of *capabilities*: deciding where perception, mapping, planning, reasoning, action generation, safety monitors, shared semantic state, and user interaction should live across CPU, GPU, NPU, MCU, memory, and interconnect resources. Modern embodied AI architectures can change where capability responsibility resides: adopting a VLA can move part of the planning-to-action function from explicit CPU-side planning logic into an accelerator-resident learned policy, dynamic VLA inference can adapt compute use at runtime (Yue et al. 2024), while agentic monitors may add CPU-heavy reasoning tasks, and safety layers may require independent low-latency access to shared state.

5.1 Provisioning consequences

Spatial composition is where the resource envelopes in Sections 3 and 4 become a placement problem. The issue is not only how much accelerator work, memory, bandwidth, or CPU orchestration a deliberative layer requires. It is which parts of that work must be isolated, which parts can be shared, which parts can be degraded, and which parts should be co-located to avoid unnecessary movement of state.

A naive architecture can reserve resources statically for every capability at its local peak. This is attractive for safety analysis because it makes interference less surprising, but it scales poorly once VLA inference, diffusion proposals, world-model updates, HRI, recovery agents, semantic monitors, and safety mechanisms each receive their own worst-case reservation. The other limiting case is dynamic placement, where only currently active capabilities consume their instantaneous requirements. This is the efficiency target, but it is not a sufficient design rule. Capabilities differ in criticality, timing band, resident state, preemptability, and tolerance to degraded service, so a safe runtime must provision between the dynamic active-set envelope and the static peak envelope while preserving isolation, latency, memory residency, and interference guarantees.

The useful spatial question is then what can be shared. Some resources should be ring-fenced: low-level control, emergency stops, runtime-assurance triggers, safety monitors, and interaction paths that must remain responsive even when deliberation is overloaded. Other work can be shared opportunistically: advisory LLM calls, explanation generation, background semantic-map refinement, non-urgent recovery planning, and low-priority imagined rollouts. A third category is shareable only when the runtime is state-aware: accelerator-resident models, KV caches, perception tensors, semantic maps, and world-model state can often be multiplexed, but moving

or duplicating them may consume more bandwidth and latency than the computation saved. This is why R^{k}_{int} is a spatial issue as much as a component-level one: dynamic placement reduces static overprovisioning only if copies, synchronization, serialization, cache movement, and shared-state arbitration do not recreate the same pressure elsewhere.

Current literature supports parts of this argument. Mixed-criticality and reservation-based scheduling explain why safety-critical and best-effort workloads cannot simply be placed in one common queue (Burns and Davis 2017; Casini et al. 2019). ROS 2 composition, tracing, and zero-copy transport show that process boundaries, callback scheduling, and data movement materially affect CPU use, latency, and analyzability (Macenski et al. 2023; Bédard et al. 2022; Bédard and Dagenais 2023; NVIDIA 2026b; Ishikawa and Kato 2025). Heterogeneous scheduling gives a language for mapping task graphs across heterogeneous processors (Topcuoglu et al. 2002; Zou et al. 2025; Fil et al. 2026). Model-serving systems show that predictable inference and high utilization require explicit control over batching, memory residency, scheduling, and KV-cache layout, rather than treating an accelerator as an undifferentiated shared device (Gujarati et al. 2020; Yu et al. 2022; Kwon et al. 2023).

Embodied.cpp treats embodied-model inference as a portable runtime problem rather than as model-specific Python glue. Its evaluation runs two VLA models in closed-loop simulation through C++ deployment paths and reports a preliminary world-action-model block benchmark with substantially reduced resident memory (Xu et al. 2026). It does not yet demonstrate a capability-aware runtime that places interacting robot workloads across heterogeneous resources (SEU-PAISys 2026). More broadly, the literature explains how to schedule callbacks, compose nodes, trace message flow, avoid copies, place kernels, serve models, or package embodied inference paths. It still falls short of answering when a semantic map, VLA policy, safety monitor, recovery agent, HRI loop, diffusion proposal generator, or world model should share a process, memory space, accelerator partition, cache, or real-time island. The forward-looking research challenge is to develop placement abstractions and runtime policies that can move robotic systems closer to the dynamic active-set envelope without giving up the reservations, isolation, analyzability, and degradation behavior that embodied systems require.

5.2 Design implications and open questions

Spatial composition is the problem of placing capabilities, not merely nodes or kernels. The static answer reserves peak resources for everything; the dynamic answer shares resources across the active set. The numerical envelopes make this a hardware and runtime issue: once deliberation reaches hundreds of GB/s to more than a TB/s of active traffic, static worst-case reservation for every capability becomes unattractive, but uncontrolled sharing can destroy latency and isolation. The central takeaway is that robotics needs capability-aware runtime

policies that can move toward dynamic sharing without giving up isolation, analyzability, safety-critical reservations, or control over data movement.

Three broad questions follow:

- Which capabilities and state should be ring-fenced, which can be shared opportunistically, and which can be shared only under explicit state-residency and criticality contracts?
- What abstractions can we use to place perception, VLA inference, diffusion, world models, safety monitors, HRI, and shared semantic state across heterogeneous compute and memory domains?
- What instrumentation shows that placement decisions improve robot behavior, not only latency, copy count, accelerator utilization, or throughput?

6 How are capabilities governed and constrained?

This question asks how robot behavior should be constrained when learned competence is not enough. The integration issue is that specifiable, safety-critical, or online requirements must become enforceable runtime objects.

Future embodied systems need governance mechanisms that are explicit, inspectable, and enforceable. This includes runtime assurance, shielded learning, control barrier functions, formal verification, safety arbitration, specification-to-control compilation, interruptibility, and safe model update paths.

This does not mean that every desired behavior should be specified symbolically or enforced by a monitor. Many behavioral requirements are vague, contextual, preference-laden, or socially learned; data-driven methods are often the right way to acquire them. The integration challenge is to support both modes. When the desired behavior is hard to state precisely, learning can provide competent default behavior. When a requirement can be stated precisely—for example a collision constraint, a temporal ordering, a workspace boundary, a force limit, a privacy rule, or a newly issued human instruction—the architecture should provide a path for that requirement to become an explicit runtime object that can constrain, monitor, or arbitrate behavior.

One useful way to understand governance is as the part of the robot architecture that turns intent and constraints into enforceable runtime structure. Control barrier functions express state constraints that can be enforced at the control layer, typically by minimally modifying nominal control inputs to preserve forward invariance of a safe set (Ames et al. 2019). Runtime assurance and Simplex-style architectures separate advanced, high-performance controllers from simpler verified backups, switching authority when the advanced controller threatens safety or violates

an assurance condition (Seto et al. 1998; Bak et al. 2014). Shielded reinforcement learning applies a related idea to learned policies: the learned component may propose actions, but a runtime shield rejects or corrects actions that would violate a formal safety condition (Alshiekh et al. 2018). Formal verification of neural-network-controlled hybrid systems extends the governance problem into learned closed-loop controllers (Ivanov et al. 2019), while model-based robotics toolchains such as Drake show how design, optimization, simulation, and verification can be brought into a shared engineering environment (Tedrake and Drake Development Team 2019). They make governance a runtime contract rather than a property hoped for after training.

Temporal logics provide a complementary route from high-level specifications to executable robot behavior. Linear temporal logic (LTL) has long been used to express sequencing, safety, liveness, recurrence, and task-completion requirements for robot controllers and supervisors (Kress-Gazit et al. 2009; Lacerda et al. 2011, 2019). Co-safe LTL and probabilistic model-checking approaches connect task specifications to policies with explicit performance or satisfaction guarantees under uncertainty (Duckworth et al. 2021; Lacerda et al. 2019). Signal temporal logic (STL) and metric temporal logic variants extend this style of specification toward real-valued signals, robustness margins, timing bounds, and continuous trajectories. This makes them especially relevant when governance must interact with MPC (Raman et al. 2014), robustness semantics for continuous signals (Fainekos and Pappas 2009), online receding-horizon control (Ilyes et al. 2025), multi-robot task allocation (Liu et al. 2025), and manipulation-specific formal methods (Zhu et al. 2026). Recent work such as robustness-to-go for STL shows that temporal-logic objectives can be reshaped to support online, receding-horizon, reactive robot control rather than only offline synthesis (Ilyes et al. 2025). Recent CBF–STL work also suggests that learning can be used to tune feasible, robust barrier-function controllers while retaining explicit temporal-logic satisfaction guarantees (Liu et al. 2025).

A related trend makes specifications differentiable, so they can participate directly in gradient-based learning, inference, and optimization. The key idea is that logic satisfaction or robustness can be represented inside the computational graph, allowing gradients to backpropagate through the specification itself into policy, trajectory, perception, or model parameters. Differentiable MPC treats an optimal-control problem as a differentiable policy layer whose costs or dynamics can be learned end-to-end (Amos et al. 2018), while differentiable nonlinear optimization libraries such as Theseus provide robotics-oriented building blocks for embedding structured least-squares problems inside learned systems (Pineda et al. 2022). On the specification side, STLCG and STLCG++ translate STL robustness into computation graphs for backpropagation and efficient parallel evaluation (Leung et al. 2023; Kapoor et al. 2025). Recent work on differentiable spatio-temporal logic extends this idea toward geometric manipulation constraints, enabling smooth spatial predicates to enter trajectory optimization and learning from demonstrations (Luo et al. 2026). This line of work is important for integration because it gives specifications a path into ML workflows: requirements can shape training losses, guide trajectory

optimization, generate constrained data, or tune policies. It should not be confused with runtime assurance, however. A differentiable penalty or smooth robustness signal helps optimize behavior, but safety-critical requirements may still need separate monitors, shields, or fallback controllers at execution time.

This line of work is directly relevant to agentic and model-maximalist robotics. As more behavior is delegated to large policies or short-lived agents, the robot needs a layer that decides what must remain invariant, what can be optimized, what can be negotiated, and what must be refused. A language agent may propose a recovery strategy, a VLA may propose an action, and a controller may preserve balance or contact stability; governance defines the arbitration rules among these proposals. Constitutional AI (Bai et al. 2022) shows one way to make behavioral principles explicit for AI systems—however, embodied robots still need those principles connected to physical state, actuation, and runtime assurance. The central integration challenge is therefore not only how to verify a controller in isolation, but how to enforce specifications across a heterogeneous stack whose components operate at different time scales, expose different forms of confidence, and may fail in qualitatively different ways.

6.1 Provisioning consequences

Governance has its own numerical burden because constraints must run at the rate of the behavior they constrain. Some checks can be sparse: policy updates, post-hoc audit, mission-level rule review, or user-facing explanations. Others must be interactive or continuous: task monitors, runtime assurance triggers, barrier-function constraints, collision checks, and controller envelopes. A governance mechanism that is correct but too slow is not a runtime governance mechanism for the behavior in question.

In the provisioning model, the most important governance effect is not merely that another monitor consumes CPU. It is that governance increases coupling. The integration term R_{int}^{k} in Equation 2 already captures overhead induced by model serving, message transport, copying, synchronization, monitoring, and shared-state access. Governance pushes this term upward because monitors, shields, runtime-assurance triggers, and arbitration logic need timely access to perception, state estimates, candidate actions, control envelopes, specifications, and fallback policies.

This effect should be read as a pressure on the ranges in Table 2, not as a separate calibrated scaling law. Adding governance can push the integration-overhead term R_{int}^{k} toward the larger values in its assumed range, and may also add direct monitor computation. Governance specifically exacerbates bandwidth requirements because it depends on timely shared-state access. While a universal governance coefficient is not known, middleware and IPC studies show that communication and copying choices can easily have tens-of-percent performance effects (Kronauer et al. 2021; Ishikawa and Kato 2025). If governance increases the bandwidth integration

coefficient by only $\Delta\eta_{\text{int}}^{\text{BW}} = 0.10$, the envelopes in Table 2 increase by tens of GB/s for interactive deliberation and by up to roughly one hundred GB/s for continuous deliberation.

Governance traffic is often latency sensitive. A safety monitor, CBF constraint, or runtime-assurance trigger cannot always be delayed like an advisory explanation or background semantic update. Explicit governance is therefore necessary, but it can make the system more concurrent, more coupled, and more bandwidth-sensitive precisely where isolation and timing guarantees matter most.

The pressure is therefore twofold. First, specifications must compile into mechanisms with the right timing band: high-level temporal logic may shape plans over seconds, STL robustness may guide receding-horizon control, and CBF-style constraints may need to run near the control layer. Second, governance must arbitrate across heterogeneous proposals. A VLA action, a recovery agent, a human command, an MPC trajectory, and a safety monitor may all speak different representational languages. The runtime needs a common way to compare authority, urgency, confidence, constraint violation, and fallback cost without turning governance into an unbounded additional workload.

6.2 Design implications and open questions

Governance is not an after-the-fact validation layer. It is part of the runtime architecture, and it must operate at the timing band of the behavior it constrains. Bandwidth sensitivity is crucial: increasing the integration-related pressure by just 10% can add tens of GB/s under interactive deliberation and up to one hundred GB/s under continuous deliberation. Future robots need mechanisms that make accurately specifiable requirements explicit and enforceable, while still allowing learned systems to handle behaviors, preferences, and social context that are difficult to specify directly.

Three broad questions follow:

- Which requirements should be learned as preferences, which should be compiled into differentiable objectives, and which must be explicit runtime constraints?
- How should governance arbitrate among learned policies, agentic planners, explicit controllers, user commands, and safety monitors when they propose conflicting actions at different rates?
- How should systems expose evidence that constraints were satisfied, violated, relaxed, overridden, or degraded during embodied execution, together with the resource cost of enforcement?

7 Implications for robot classes

The provisioning argument becomes more concrete when mapped back to embodied platforms. Different robot designs expose different coupling patterns, timing constraints, power budgets, safety cases, sensing suites, and task values; these differences determine how much deliberation is worth paying for. The three deliberation levels are therefore design regimes, not a ladder that every robot class must climb. Advisory deliberation may be the right endpoint for a cost-constrained consumer robot; interactive deliberation may be the useful regime for quadrupeds, cobots, and many mobile manipulators; continuous deliberation may be justified only when the task value of persistent world modeling and shared semantic state outweighs the cost in memory residency, bandwidth, orchestration, and governance.

Table 3 gives a qualitative mapping from robot classes to plausible deliberative regimes. It combines the provisioning envelopes in Table 2 with a coupling-oriented view of robot classes: different embodiments place different pressure on perception, control, manipulation, locomotion, safety, user interaction, and shared state. The combination appears in two places: the selected deliberative regime reflects how much advisory, interactive, or continuous deliberation a class can plausibly justify, while the final column identifies the barriers to moving toward the next regime. The mapping is intentionally approximate: actual requirements depend on task, payload, sensing suite, power and thermal envelope, safety case, user interaction, and whether deliberation is local, remote, or split across the system. Its purpose is to close the loop from abstract resource terms back to concrete robot designs.

Table 3. Qualitative mapping from robot classes to deliberation regimes.

Robot class	**Plausible deliberative regime**	**What it enables**	**Barriers toward next regime**
Consumer domestic robots	Advisory deliberation	Scene interpretation, user instruction handling, exception explanation, and occasional recovery advice.	Cost, power, privacy-sensitive perception, limited resident memory, and low bandwidth headroom.
Drones	Advisory deliberation	Mission-level semantic advice, target interpretation, and non-real-time recovery or replanning.	Weight, power, flight-control isolation, and hard real-time safety margins.
Industrial arms	Advisory deliberation	Programming assistance, task setup, anomaly explanation, and guarded manipulation updates.	Certification, deterministic control, safety case stability, and limited need for continuous semantic reasoning.

Cobots	Advisory/interactive deliberation	Human-facing task setup, guarded collaboration, manipulation recovery, and semantic interpretation of workcell context.	Safety certification, human proximity, predictable interaction, and bounded adaptation around collaborative manipulation.
AMRs	Advisory/interactive deliberation	Navigation exceptions, fleet interaction, human-facing explanations, and semantic routing.	Perception/navigation bandwidth, safety-certified behavior, and interaction with fleet/runtime infrastructure.
Quadrupeds	Interactive deliberation	Terrain understanding, recovery planning, semantic inspection, and user-guided task adaptation.	Coordinating semantic reasoning with fast locomotion, balance recovery, and shared state without disrupting control.
Mobile manipulators	Interactive/continuous deliberation	Manipulation while moving, whole-body reaching, dynamic object handling, moving-conveyor interaction, throwing/catching, and contact-rich recovery.	Coordinating mobility, manipulation, perception, full-body control, safety monitoring, and shared semantic/geometric state under tight bandwidth and timing constraints.
Humanoids	Interactive/continuous deliberation	Whole-body task reasoning, persistent world state, rich HRI, manipulation recovery, and multi-capability arbitration.	Co-resident models, world state, safety monitors, high-bandwidth shared memory, and orchestration across tightly coupled capabilities.

This mapping helps interpret the numerical envelopes. For a consumer vacuum cleaner or simple domestic robot, advisory deliberation is interesting because it can extend the robot's useful remit beyond its nominal task. A vacuum cleaner already traverses the home with sensors; a semantic layer could help search for displaced items, notice that a missing sock is under a bed, support lightweight household surveillance, explain cleaning failures, or interpret user instructions that refer to objects and places. The hard part is making that capability cheap, private, power-efficient, and robust enough that it does not disturb the simpler control and navigation stack. In this regime, the important question is not whether a large model can solve the semantic task, but whether the robot can afford the resident memory, sensor-to-model bandwidth, and integration wrapper needed to use it occasionally and safely.

Quadrupeds, cobots, mobile manipulators, and simpler humanoids make the case for interactive deliberation more naturally, but for different reasons. Cobots emphasize guarded collaboration, predictable human interaction, and manipulation recovery within structured workcells. Mobile manipulators add a stronger coupling problem: locomotion, manipulation, perception, and whole-body control may all be active while the robot catches or throws objects, operates on moving conveyors, manipulates while driving, or recovers from contact-rich failures. The provisioning pressure shifts from occasional model invocation to scheduling: VLA or VLM inference, diffusion-style action proposals, perception, control, and safety monitors must coexist without starving latency-sensitive functions. This is where the Section 4 bandwidth ranges may go beyond abstract estimates and become design guidance.

Continuous deliberation is most compelling for humanoids and highly capable mobile manipulation systems, but it is also where the integration thesis becomes most severe. Persistent world models, shared semantic state, multi-agent reasoning, rich HRI, and runtime assurance all compete for resident memory and bandwidth. The design question is therefore not simply how to put a larger model on the robot. It is how to decide which capabilities should be continuous, which should be interactive, which should remain advisory, and which should be ring-fenced for safety and responsiveness.

8 Conclusions and outlook

Robotics is becoming a systems discipline of coupled capabilities, not a sequence of isolated perception, planning, and control modules. The future of robotics will not be determined only by larger models or isolated algorithmic advances. More capable robots require systems that realize capabilities through appropriate paradigms, compose them across time, distribute them across heterogeneous compute, and govern them under explicit constraints.

In this view, robotics becomes an integration-centric systems science: a discipline concerned with how heterogeneous capabilities are made to work together under latency, bandwidth, memory, power, uncertainty, and safety constraints. The path forward is not a single architecture, model class, or runtime. It is a research program around integrated workloads: which capabilities are active together, how much state they share, how quickly that state must be refreshed, where computation should reside, and which runtime mechanisms can interrupt, validate, or override behavior. Looking beyond these immediate systems questions, the discussion below zooms out to five broader themes likely to shape robotics research: coupling and bandwidth, semantic quality of service, shifting composition boundaries, governance beyond training, and scaling laws for embodied competence.

Coupling and bandwidth

Coupling should be treated as a first-class systems variable. As robot capabilities become more mutually dependent, they increase concurrency, shared-state interaction, bandwidth demand, CPU coordination overhead, and orchestration complexity. Transformer serving, diffusion action generation, and world-model updates all create repeated access to model weights, state, history, and shared latent representations; systems work on attention and LLM serving already shows the importance of IO-awareness, KV-cache management, and memory scheduling (Dao et al. 2022; Kwon et al. 2023). Robotics compounds this with closed-loop deadlines and shared physical state. A key path forward is therefore to measure and design around co-active workloads, not only peak model throughput.

Semantic QoS

Robot cognition increasingly resembles a distributed inference system maintaining shared semantic, geometric, task, and safety state under latency constraints. We use *semantic quality of service* to denote runtime-visible properties of semantic outputs: freshness, confidence, task relevance, spatial precision, uncertainty, deadline behavior, and degradation mode. The term is related to work in semantic and task-oriented communications, where communication systems are evaluated not only by bits delivered, but by whether transmitted information is useful for the receiver's task (Gündüz et al. 2023). For robotics, the research challenge is to make semantic state schedulable, inspectable, and governable: a runtime should know when "the cup is graspable," "this is an unsafe scene," or "the instruction refers to the red tool" is fresh enough, reliable enough, and relevant enough to affect action.

Moving composition boundaries

The boundary between planning, control, and reasoning is becoming less architectural and more negotiated at runtime. Diffusion policies turn action generation into iterative conditional inference (Chi et al. 2025); model-predictive control keeps optimization inside the execution loop (Mayne 2014); LLM planners and code policies can generate task structure on demand (Singh et al. 2023; Liang et al. 2023); and VLA policies attempt to produce action directly from multimodal context (Brohan et al. 2023; Kim et al. 2024). Explicit composition, agentic composition, and model-maximalism are all reasonable possibilities. The path forward is to strike a balance between explicit runtime structure, dynamic agentic deliberation, and learned end-to-end behavior while preserving latency, resource predictability, safety, and debuggability.

Governance beyond training

Training-time alignment, dataset filtering, and offline evaluation are essential for shaping default behavior, especially when desired behavior is difficult to specify directly. They are not enough, however, for requirements that must be introduced online, audited, interrupted, arbitrated, or

enforced at the moment of action. Robots need governance mechanisms that remain external enough to inspect and enforce when explicit constraints are available, yet integrated enough to operate at the relevant timing band. Runtime assurance, shields, barrier functions, temporal-logic specifications, differentiable specifications, and safety monitors should therefore be studied as part of the systems architecture, including their bandwidth, state-access, and orchestration costs.

Scaling laws for embodied competence

Language-model studies relate loss predictably to model size, data, and training compute (Kaplan et al. 2020). Chinchilla uses those relationships to identify compute-optimal training (Hoffmann et al. 2022). Bounded robotics studies now report related regularities. Waymo connects training-compute scaling to improved motion forecasting and closed-loop planning metrics (Baniodeh et al. 2025). DeepFleet derives architecture-specific model/data scaling for warehouse fleets (Agaskar et al. 2025). Manipulation experiments show that environment and object diversity can matter more than repeated demonstrations (Lin et al. 2024). However, these results do not establish a universal law for robotics. Useful laws will need to hold the task, model family, architecture, and evaluation protocol fixed; distinguish trajectory volume from task, environment, object, and embodiment diversity; and relate training loss to closed-loop success, failure tails, intervention, cycle time, safety, and resource cost. The likely outcome is a family of task- and architecture-specific laws that guide both model scaling and robot-brain provisioning.

Appendix: Provisioning relations

This appendix collects the back-of-the-envelope relations used to construct the component anchors in Table 1. They are intended as transparent accounting relations rather than calibrated performance models. The auxiliary terms below make explicit the assumptions about token counts, active fractions, denoising steps, proposal rates, rollout branching, resident state, and memory traffic. Tables 4 and 5 list the exact working ranges from which the rounded values in Sections 3.1 and 4.1 are chosen.

Table 4. Working component-level ranges used to construct the rounded anchors in Table 1. Accelerator values are FP8-equivalent TFLOP of burst work per active episode, proposal, or rollout bundle. Memory values are resident GB. Bandwidth values are active-phase GB/s where a rate assumption is needed.

Component	Accelerator range	Memory range	Bandwidth range	CPU term
Autoregressive VLM/VLA	0.36–1.44 TFLOP for a 3B advisor; 2.2–8.4 TFLOP for a 7B reasoner	6–9 GB for compact 3B advisory use; 12–20 GB for 7B-class use with KV and vision/runtime buffers	300–400 GB/s for an interactive 7B-class model; 400–600 GB/s for richer near-continuous use	Prompt assembly, context packaging, serving dispatch, tool-mediated control flow
Diffusion action generator	1.6–7.2 TFLOP per compact action proposal	0.1–4 GB depending on denoiser size, visual encoder, features, and scratch space	40–120 GB/s for 4–6 denoising steps at 5–10 Hz with repeated state access	Proposal scheduling, receding-horizon handoff, trajectory-buffer management
World model	1.1–18 TFLOP per rollout/update bundle	0.5–8 GB for weights, latent state, branches, history, and runtime buffers	8–150 GB/s for 1–3 branches at 2–5 Hz with latent/history traffic	Rollout management, branch selection, state-history bookkeeping
Integration overhead	0.15–5.04 TFLOP for the co-active ranges above (η = 5–15%)	0.66–9.6 GB for the co-active ranges above (η = 10–30%)	34.8–261 GB/s for the co-active ranges above (η = 10–30%)	Message transport, copying, synchronization, monitoring, shared-state access

Table 5. Working temporal provisioning ranges used to construct Table 2. Accelerator values multiply per-episode work from Table 4 by invocation or update rate and are therefore sustained FP8-equivalent TFLOP/s. Total rows add all co-active component and integration terms.

Resource	**Component**	**Advisory deliberation**	**Interactive deliberation**	**Continuous deliberation**
Assumptions	AR model size	3B parameters	7B parameters	7B parameters
	Diffusion model	—	0.1–4 GB resident	0.1–4 GB resident
	World-model size	—	—	50M–400M parameters
	AR invocation rate	0.1–1 Hz	1–5 Hz	5–20 Hz
	AR active fraction	0.15–0.25	0.5–0.7	0.7–1.0
	Diffusion proposal rate	—	5–10 Hz	10–20 Hz
	World update/branching	—	—	2–5 Hz, 1–3 branches
Accelerator, FP8-equiv. TFLOP/s	R_{AR}	0.036–1.44	2.2–42	11–168
	R_{diff}	—	8–72	16–144
	R_{world}	—	—	2.2–90
	R^{k}_{int} (η = 5–15%)	0.002–0.216	0.51–17.1	1.46–60.3
	Total	**0.038–1.66**	**10.7–131**	**30.7–462**
Memory (GB)	R_{AR}	4–12	12–20	18–30
	R_{diff}	—	0.1–4	0.1–4
	R_{world}	—	—	0.5–8
	R^{k}_{int} (η = 10–30%)	0.4–3.6	1.21–7.2	1.86–12.6
	Total	**4–16**	**13–31**	**21–55**
Bandwidth (GB/s)	R_{AR}	45–75	150–280	280–600
	R_{diff}	—	40–120	80–240
	R_{world}	—	—	20–150
	R^{k}_{int} (η = 10–30%)	4.5–22.5	19–120	38–297
	Total	**50–98**	**210–520**	**418–1287**

CPU	Threads/processes	Episodic request handlers and serving dispatch.	Concurrent agent, tool, and serving processes.	Persistent queues, monitors, and arbitration processes.
	Control-plane work	Prompt assembly and sensor/context packaging.	Retrieval, serialization, routing, and callback scheduling.	Shared-state access, monitor coordination, and contention-induced jitter.

For autoregressive VLM/VLA inference, accelerator work per active episode scales directionally as

$$R_{\mathrm{AR}}^{\mathrm{accel}} \approx 2P_{\mathrm{AR}}n_{\mathrm{tok}}h_{\mathrm{AR}} \qquad (4),$$

following the standard transformer scaling relation between active parameters and processed tokens (Kaplan et al. 2020). Here P_{AR} is active parameter count, n_{tok} is the token budget for the active episode, including generated tokens and effective prefill/multimodal tokens, and h_{AR} captures additional modeled work from multimodal encoding and operations omitted by the simple token-scaling relation. It does not compensate for achieved-hardware utilization. Plugging in a compact 3B advisory model with $n_{\mathrm{tok}} = 40$–120 and $h_{\mathrm{AR}} = 1.5$–2 gives roughly 0.36–1.44 FP8-equivalent TFLOP per advisory episode. A 7B interactive model with $n_{\mathrm{tok}} = 80$–200 and $h_{\mathrm{AR}} = 2$–3 gives roughly 2.2–8.4 TFLOP per active episode.

Autoregressive memory contains weights, KV cache, vision inputs, and runtime buffers:

$$R_{\mathrm{AR}}^{\mathrm{mem}} = M_{\mathrm{AR}}^{\mathrm{weights}} + M_{\mathrm{AR}}^{\mathrm{KV}} + M_{\mathrm{AR}}^{\mathrm{vision}} + M_{\mathrm{AR}}^{\mathrm{runtime}} \qquad (5).$$

The weight term is $M_{\mathrm{AR}}^{\mathrm{weights}} \approx P_{\mathrm{AR}}q_{\mathrm{AR}}$, where q_{AR} is the number of bytes per parameter after quantization. KV-cache behavior is governed by context length, layers, hidden dimension, and serving strategy (Kwon et al. 2023). For a 7B-class model with 8-bit weights (Dettmers et al. 2022; Frantar et al. 2023), the weight footprint is about 7 GB. With an 8192-token context, roughly 32 layers, hidden size 4096, and FP16 key/value elements (Kwon et al. 2023), a simple KV-cache estimate $2 \times 8192 \times 32 \times 4096 \times 2$ bytes gives about 4.3 GB. The weights and KV cache therefore contribute about 11.3 GB before vision buffers and runtime scratch space. For a compact 3B advisor with 8-bit weights, the weights alone contribute about 3 GB; a shorter 4096-token context with roughly 32 layers and hidden size 3072 gives a KV-cache estimate of about 1.6 GB, so weights and KV cache contribute about 4.6 GB before vision buffers and runtime scratch space.

As a coarse approximation, sustained autoregressive bandwidth can be treated as proportional to duty factor, bytes moved per event, and events per second:

$$R_{\mathrm{AR}}^{\mathrm{BW}} \approx d_{\mathrm{AR}} h_K \left(B_{\mathrm{AR}}^{\mathrm{tok}} r_{\mathrm{tok}} + B_{\mathrm{AR}}^{\mathrm{req}} f_{\mathrm{req}} + B_{\mathrm{AR}}^{\mathrm{vision}} f_{\mathrm{vision}} \right) \qquad (6).$$

Here $d_{\mathrm{AR}} \in [0,1]$ is the duty factor over which the component is active or must be provisioned as active, h_K captures additional modeled data movement from burstiness, concurrency, rereads, and copies, $B_{\mathrm{AR}}^{\mathrm{tok}}$ is per-token memory movement for decode, r_{tok} is the active-phase token rate, $B_{\mathrm{AR}}^{\mathrm{req}}$ is per-request movement for prompt prefill, context setup, and runtime buffers, f_{req} is the active-phase request rate, and $B_{\mathrm{AR}}^{\mathrm{vision}} f_{\mathrm{vision}}$ captures the active-phase movement of vision tensors and other multimodal inputs. Batch-1 autoregressive decode repeatedly streams model weights and active KV state (Kwon et al. 2023; Chen 2026). For a 7B-class model with 8-bit weights (Dettmers et al. 2022; Frantar et al. 2023), weight traffic alone is approximately $P_{\mathrm{AR}} q_{\mathrm{AR}} \approx 7$ GB per decoded token. An interactive target of $r_{\mathrm{tok}} = 40$–50 tokens/s therefore gives about 280–350 GB/s from weight traffic alone, before KV-cache traffic, activations, prompt setup, vision tensors, and $h_K > 1$ are included.

A diffusion action generator scales approximately as

$$R_{\mathrm{diff}}^{\mathrm{accel}} \approx C_{\mathrm{step}} N_{\mathrm{step}} h_{\mathrm{diff}} \qquad (7),$$

because each proposal requires multiple denoising steps (Ho et al. 2020; Song et al. 2021). Here C_{step} is the accelerator work per denoising step, N_{step} is the number of denoising steps per action proposal, and h_{diff} captures additional modeled work from conditioning, encoding, and operations omitted by the per-step anchor. It does not compensate for achieved-hardware utilization. With $C_{\mathrm{step}} = 0.2$–0.4 TFLOP, $N_{\mathrm{step}} = 4$–6 (Chi et al. 2025), and $h_{\mathrm{diff}} = 2$–3, a compact action proposal contributes roughly 1.6–7.2 FP8-equivalent TFLOP of accelerator work before temporal repetition is considered.

The corresponding resident-memory term includes the diffusion model, conditioning inputs, action or trajectory buffers, and runtime scratch space:

$$R_{\mathrm{diff}}^{\mathrm{mem}} \approx M_{\mathrm{diff}}^{\mathrm{weights}} + M_{\mathrm{diff}}^{\mathrm{cond}} + M_{\mathrm{diff}}^{\mathrm{traj}} + M_{\mathrm{diff}}^{\mathrm{runtime}} \qquad (8).$$

Chi et al. (2025) anchor this decomposition through visual conditioning, receding-horizon action generation, and iterative denoising over action sequences. The public image-conditioned training configuration (Real-Stanford Diffusion Policy Contributors 2023) uses a horizon of 16, two observation steps, eight action steps, 100 inference steps, a ResNet-18 visual encoder, and U-Net channel widths of 512, 1024, and 2048. The raw action trajectory itself is small: a 16-step 7-DoF FP16 action trajectory is well below 1 KB. Two FP16 RGB crops of size 76×76 occupy only about 0.07 MB per camera stream. By contrast, the ResNet-18 encoder alone is listed by Torchvision as 11,689,512 parameters with a 44.7 MB FP32 checkpoint footprint (PyTorch Contributors 2026), or roughly 23 MB in FP16 before the diffusion denoiser, feature maps, activations, batching buffers, and runtime scratch space are included. A compact policy can

plausibly sit in the 0.1–1 GB range, while richer visual conditioning, larger denoisers, batching buffers, and co-resident feature maps can push the term toward 1–4 GB.

We do not assign this diffusion policy a single parameter count: the cited configuration specifies its encoder and U-Net widths, while the provisioning calculation is parameterized by work per denoising step and resident memory. The 0.1–4 GB range is therefore a total resident envelope, not a claim about diffusion-model weight size. Table 2 uses the upper work and residency points for this architecture and changes its proposal rate, not its assumed model size, between interactive and continuous deliberation.

The corresponding state-movement pressure can be approximated as

$$R_{\text{diff}}^{\text{BW}} \approx B_{\text{diff}}^{\text{state}} N_{\text{step}} f_{\text{diff}} h_K \qquad (9),$$

where $B_{\text{diff}}^{\text{state}}$ is the data moved per denoising step to read and write robot state, perception features, task context, and trajectory buffers; N_{step} is the number of denoising steps per proposal; f_{diff} is the proposal rate; and h_K is the burst/concurrency factor defined above. This term reflects the repeated state-conditioned denoising used by diffusion policies and diffusion planners (Chi et al. 2025; Janner et al. 2022). Plugging in $B_{\text{diff}}^{\text{state}} = 1$ GB, $N_{\text{step}} = 4$–6, $f_{\text{diff}} = 5$–10 Hz, and $h_K = 2$ gives roughly 40–120 GB/s for an interactive action generator.

World models add rollout length and branching:

$$R_{\text{world}}^{\text{accel}} \approx C_{\text{update}} N_{\text{rollout}} N_{\text{branch}} h_{\text{world}} \qquad (10),$$

where C_{update} is work per model update or rollout step, N_{rollout} is rollout length, N_{branch} is the number of imagined futures or rollout branches (Hafner et al. 2023; Micheli et al. 2023; Zidan et al. 2026), and h_{world} captures additional modeled work from prediction heads, branch management, and operations omitted by the per-update anchor, rather than achieved-hardware utilization. With $C_{\text{update}} = 0.05$–0.2 TFLOP, $N_{\text{rollout}} = 15$, $N_{\text{branch}} = 1$–3, and $h_{\text{world}} = 1.5$–2, an active rollout bundle contributes roughly 1.1–18 TFLOP.

Resident memory grows with the world model, latent state, branch buffers and temporal history (Hafner et al. 2023; Micheli et al. 2023):

$$R_{\text{world}}^{\text{mem}} \approx M_{\text{world}}^{\text{weights}} + M_{\text{world}}^{\text{latent}} + N_{\text{branch}} M_{\text{world}}^{\text{branch}} + M_{\text{world}}^{\text{history}} + M_{\text{world}}^{\text{runtime}} \qquad (11).$$

Here $M_{\text{world}}^{\text{latent}}$ is the current latent state, $M_{\text{world}}^{\text{branch}}$ is the memory retained per imagined rollout branch, $M_{\text{world}}^{\text{history}}$ captures temporal context or replay/state history needed for prediction, and $M_{\text{world}}^{\text{runtime}}$ covers feature tensors, activations, batching buffers, and scratch space. DreamerV3's public configuration exposes model-size families from 1M to 400M parameters and default imagined rollout length 15 (Hafner 2025). A 50M–400M-parameter world model has a raw FP16

weight footprint of roughly 0.1–0.8 GB. As a concrete provisioning envelope, we use $M_{\text{world}}^{\text{latent}} = 0.05–0.5$ GB, $M_{\text{world}}^{\text{branch}} = 0.1–0.7$ GB per retained rollout branch with $N_{\text{branch}} = 1–3$, $M_{\text{world}}^{\text{history}} = 0.1–1$ GB, and $M_{\text{world}}^{\text{runtime}} = 0.15–3.7$ GB. Together with the weight term, these assumptions give roughly $R_{\text{world}}^{\text{mem}} = 0.5–8$ GB.

The corresponding world-model bandwidth pressure can be approximated as

$$R_{\text{world}}^{\text{BW}} \approx B_{\text{world}}^{\text{latent}} N_{\text{branch}} f_{\text{world}} h_K \qquad (12),$$

where $B_{\text{world}}^{\text{latent}}$ is the data moved per world-model update to read and write latent state, branch buffers, and relevant history; N_{branch} is the number of imagined futures or rollout branches; f_{world} is the update rate; and h_K is again the burst/concurrency factor. We set $N_{\text{branch}} = 1–3$ to represent single-trajectory prediction through modest branched evaluation, and $f_{\text{world}} = 2–5$ Hz to represent an interactive 200–500 ms deliberative update cadence. The traffic term $B_{\text{world}}^{\text{latent}} = 2–5$ GB is derived from the DreamerV3 resident-state envelope above by assuming that each update rereads and rewrites a substantial fraction of latent state, branch buffers, temporal history, and runtime feature tensors. With $h_K = 2$ as an illustrative burst/concurrency factor, these assumptions give roughly 8–150 GB/s, with the upper end corresponding to branched receding-horizon prediction or safety evaluation.

Equation 2 then applies a sensitivity factor to account for model serving, adaptation layers, message transport, copying, serialization, monitoring, synchronization, and shared-state access. The coefficient should be interpreted as a design sensitivity, not as a universal constant.

References

Agaskar, Ameya, Sriram Siva, William Pickering, et al. 2025. *DeepFleet: Multi-Agent Foundation Models for Mobile Robots*. arXiv preprint arXiv:2508.08574. https://arxiv.org/abs/2508.08574.

Ahn, Michael, Anthony Brohan, Noah Brown, et al. 2022. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. arXiv preprint arXiv:2204.01691. https://arxiv.org/abs/2204.01691.

Ahn, Michael, Debidatta Dwibedi, Chelsea Finn, et al. 2024. *AutoRT: Embodied Foundation Models for Large Scale Orchestration of Robotic Agents*. arXiv preprint arXiv:2401.12963. https://arxiv.org/abs/2401.12963.

Alshiekh, Mohammed, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. "Safe Reinforcement Learning via Shielding." *AAAI Conference on Artificial Intelligence*.

Ames, Aaron D., Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. 2019. "Control Barrier Functions: Theory and Applications." *European Journal of Control* 49: 1–21.

Amos, Brandon, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J. Zico Kolter. 2018. "Differentiable MPC for End-to-End Planning and Control." *Advances in Neural Information Processing Systems*.

Bai, Shuai, Keqin Chen, Xuejing Liu, et al. 2025. *Qwen2.5-VL Technical Report*. arXiv preprint arXiv:2502.13923. https://arxiv.org/abs/2502.13923.

Bai, Yuntao, Saurav Kadavath, Sandipan Kundu, et al. 2022. *Constitutional AI: Harmlessness from AI Feedback*. arXiv preprint arXiv:2212.08073. https://arxiv.org/abs/2212.08073.

Bak, Stanley, Taylor T. Johnson, Marco Caccamo, and Lui Sha. 2014. "Real-Time Reachability for Verified Simplex Design." 2014 IEEE Real-Time Systems Symposium (RTSS), 138–148.

Baniodeh, Mustafa, Kratarth Goel, Scott Ettinger, et al. 2025. *Scaling Laws of Motion Forecasting and Planning—a Technical Report*. arXiv preprint arXiv:2506.08228. https://arxiv.org/abs/2506.08228.

Bédard, Christophe, and Michel Dagenais. 2023. "Message Flow Analysis with Complex Causal Links for Distributed ROS 2 Systems." *Robotics and Autonomous Systems* 161: 104361. https://doi.org/10.1016/j.robot.2022.104361.

Bédard, Christophe, Ingo Lütkebohle, and Michel Dagenais. 2022. "ros2_tracing: Multipurpose Low-Overhead Framework for Real-Time Tracing of ROS 2." *IEEE Robotics and Automation Letters* 7 (3): 6511–18. https://doi.org/10.1109/LRA.2022.3174346.

Beyer, Lucas, Andreas Steiner, André Susano Pinto, et al. 2024. *PaliGemma: A Versatile 3B VLM for Transfer*. arXiv preprint arXiv:2407.07726. https://arxiv.org/abs/2407.07726.

Björck, Johan, Fernando Castañeda, Nikita Cherniadev, et al. 2025. *GR00T N1: An Open Foundation Model for Generalist Humanoid Robots*. arXiv preprint arXiv:2503.14734. https://arxiv.org/abs/2503.14734.

Black, Kevin, Noah Brown, Danny Driess, et al. 2025. "π_0: A Vision-Language-Action Flow Model for General Robot Control." *Robotics: Science and Systems*.

Bonasso, R. Peter, R. James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Marc G. Slack. 1997. "Experiences with an Architecture for Intelligent, Reactive Agents." *Journal of Experimental & Theoretical Artificial Intelligence* 9 (2–3): 237–56.

Brohan, Anthony, Noah Brown, Justice Carbajal, et al. 2023. *RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control*. arXiv preprint arXiv:2307.15818. https://arxiv.org/abs/2307.15818.

Brooks, Rodney A. 1986. "A Robust Layered Control System for a Mobile Robot." *IEEE Journal on Robotics and Automation* 2 (1): 14–23.

Burns, Alan, and Robert I. Davis. 2017. "A Survey of Research into Mixed Criticality Systems." *ACM Computing Surveys* 50 (6): 82:1–37. https://doi.org/10.1145/3131347.

Carvalho, João, An Thai Le, Piotr Kicki, Dorothea Koert, and Jan Peters. 2025. "Motion Planning Diffusion: Learning and Adapting Robot Motion Planning with Diffusion Models." *IEEE Transactions on Robotics* 41: 4881–901. https://doi.org/10.1109/TRO.2025.3593109.

Casini, Daniel, Tobias Blass, Ingo Lütkebohle, and Björn B. Brandenburg. 2019. "Response-Time Analysis of ROS 2 Processing Chains Under Reservation-Based Scheduling." *31st Euromicro Conference on Real-Time Systems*.

Chen, Josef. 2026. *Memory-Bound but Not Bandwidth-Limited: The Physical AI Inference Gap in Batch-1 LLM Decode*. arXiv preprint arXiv:2605.30571. https://arxiv.org/abs/2605.30571.

Chen, Lili, Kevin Lu, Aravind Rajeswaran, et al. 2021. "Decision Transformer: Reinforcement Learning via Sequence Modeling." *Advances in Neural Information Processing Systems*.

Chi, Cheng, Zhenjia Xu, Siyuan Feng, et al. 2025. “Diffusion Policy: Visuomotor Policy Learning via Action Diffusion.” *The International Journal of Robotics Research* 44 (10–11): 1684–704. https://doi.org/10.1177/02783649241273668.

Colledanchise, Michele, and Petter Ögren. 2018. *Behavior Trees in Robotics and AI: An Introduction*. CRC Press.

Dao, Tri, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness.” *Advances in Neural Information Processing Systems*.

Dettmers, Tim, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. “LLM.int8(): 8-Bit Matrix Multiplication for Transformers at Scale.” *Advances in Neural Information Processing Systems*.

Driess, Danny, Fei Xia, Mehdi S. M. Sajjadi, et al. 2023. *PaLM-E: An Embodied Multimodal Language Model*. arXiv preprint arXiv:2303.03378. https://arxiv.org/abs/2303.03378.

Duckworth, Paul, Bruno Lacerda, and Nick Hawes. 2021. “Time-Bounded Mission Planning in Time-Varying Environments.” *IEEE Robotics and Automation Letters* 6 (2): 2035–42.

Fainekos, Georgios E., and George J. Pappas. 2009. “Robustness of Temporal Logic Specifications for Continuous-Time Signals.” *Theoretical Computer Science* 410 (42): 4262–91.

Fil, Jakub, Yulia Sandamirskaya, Hector Andres Gonzalez, et al. 2026. *Heterogeneous Computing Platform for Real-Time Robotics*. arXiv preprint arXiv:2601.09755. https://arxiv.org/abs/2601.09755.

Frantar, Elias, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. “GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers.” *International Conference on Learning Representations*.

Gujarati, Arpan, Reza Karimi, Safya Alzayat, et al. 2020. “Serving DNNs Like Clockwork: Performance Predictability from the Bottom Up.” *14th USENIX Symposium on Operating Systems Design and Implementation*, 443–62.

Gündüz, Deniz, Zhijin Qin, Inaki Estella Aguerri, et al. 2023. “Beyond Transmitting Bits: Context, Semantics, and Task-Oriented Communications.” *IEEE Journal on Selected Areas in Communications* 41 (1): 5–41. https://doi.org/10.1109/JSAC.2022.3223408.

Guo, Jun, Piaopiao Jin, Jason Li, et al. 2026. *Xiaomi-Robotics-1: Scaling Vision-Language-Action Models with over 100K Hours of Real-World Trajectories*. arXiv preprint arXiv:2607.15330. https://arxiv.org/abs/2607.15330.

Hafner, Danijar. 2025. *DreamerV3 Public Configuration File*. Https://raw.githubusercontent.com/danijar/dreamerv3/main/dreamerv3/configs.yaml.

Hafner, Danijar, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2023. *Mastering Diverse Domains Through World Models*. arXiv preprint arXiv:2301.04104. https://arxiv.org/abs/2301.04104.

Ho, Jonathan, Ajay Jain, and Pieter Abbeel. 2020. “Denoising Diffusion Probabilistic Models.” *Advances in Neural Information Processing Systems*.

Hoffmann, Jordan, Sebastian Borgeaud, Arthur Mensch, et al. 2022. *Training Compute-Optimal Large Language Models*. arXiv preprint arXiv:2203.15556. https://arxiv.org/abs/2203.15556.

Hu, Guoqiang, Wee Peng Tay, and Yonggang Wen. 2012. “A Survey of Cloud Robotics and Automation.” *IEEE International Conference on Robotics and Biomimetics*, 1248–53.

Huang, Wenlong, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. 2023. “VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models.” *Conference on Robot Learning*.

Huang, Wenlong, Fei Xia, Ted Xiao, et al. 2023. *Inner Monologue: Embodied Reasoning Through Planning with Language Models*. arXiv preprint arXiv:2207.05608. https://arxiv.org/abs/2207.05608.

Ichter, Brian, Anthony Brohan, Yevgen Chebotar, et al. 2023. “Do As I Can, Not As I Say: Grounding Language in Robotic Affordances.” In Proceedings of the 6th Conference on Robot Learning, Proceedings of Machine Learning Research 205: 287–318. PMLR.

Ilyes, Roland, Lara Brudermüller, Bruno Lacerda, and Nick Hawes. 2025. *Ro-to-Go: Robustness-to-Go for Signal Temporal Logic in Receding Horizon Control*. arXiv preprint arXiv:2504.20887. https://arxiv.org/abs/2504.20887.

Ishikawa, Yuki, and Shinpei Kato. 2025. *Agnocast: A Framework for Fast IPC in ROS 2 Applications*. arXiv preprint arXiv:2505.00836. https://arxiv.org/abs/2505.00836.

Ivanov, Radoslav, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. 2019. “Verisig: Verifying Safety Properties of Hybrid Systems with Neural Network Controllers.” *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, 169–78.

Janner, Michael, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. 2022. “Planning with Diffusion for Flexible Behavior Synthesis.” *International Conference on Machine Learning*.

Jiang, Jevin, Ying Chen, Blake A. Hechtman, Fenghui Zhang, and Yarong Mu. 2026. *Ragged Paged Attention: A High-Performance and Flexible LLM Inference Kernel for TPU*. arXiv preprint arXiv:2604.15464. https://arxiv.org/abs/2604.15464.

Kaplan, Jared, Sam McCandlish, Tom Henighan, et al. 2020. *Scaling Laws for Neural Language Models*. arXiv preprint arXiv:2001.08361. https://arxiv.org/abs/2001.08361.

Kapoor, Parv, Kazuki Mizuta, Eunsuk Kang, and Karen Leung. 2025. *STLCG++: A Masking Approach for Differentiable Signal Temporal Logic Specification*. arXiv preprint arXiv:2501.04194. https://arxiv.org/abs/2501.04194.

Kehoe, Ben, Sachin Patil, Pieter Abbeel, and Ken Goldberg. 2015. "A Survey of Research on Cloud Robotics and Automation." *IEEE Transactions on Automation Science and Engineering* 12 (2): 398–409.

Kim, Moo Jin, Karl Pertsch, Siddharth Karamcheti, et al. 2024. *OpenVLA: An Open-Source Vision-Language-Action Model*. arXiv preprint arXiv:2406.09246. https://arxiv.org/abs/2406.09246.

Kress-Gazit, Hadas, Georgios E. Fainekos, and George J. Pappas. 2009. "Temporal-Logic-Based Reactive Mission and Motion Planning." *IEEE Transactions on Robotics* 25 (6): 1370–81.

Kronauer, Andreas, Tim Pohlmann, and Richard Matthaei. 2021. "Latency Analysis of ROS2 Multi-Node Systems." *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 1–7. https://doi.org/10.1109/MFI52462.2021.9591166.

Kwon, Woosuk, Zhuohan Li, Siyuan Zhuang, et al. 2023. "Efficient Memory Management for Large Language Model Serving with PagedAttention." *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 611–26.

Lacerda, Bruno, Fahad Faruq, David Parker, and Nick Hawes. 2019. "Optimal Policy Generation for Partially Satisfiable Co-Safe LTL Specifications." *International Journal of Robotics Research* 38 (7): 781–94.

Lacerda, Bruno, David Parker, and Nick Hawes. 2011. "Optimal and Dynamic Planning for Markov Decision Processes with Co-Safe LTL Specifications." *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1511–16.

Lajoie, Pierre-Yves, Christophe Bédard, Giovanni Beltrame, and Michel Dagenais. 2022. *Analyze, Debug, Optimize: Real-Time Tracing for Perception and Mapping Systems in Robotics*. arXiv preprint arXiv:2204.11778. https://arxiv.org/abs/2204.11778.

LaValle, Steven M. 2006. *Planning Algorithms*. Cambridge University Press.

Leung, Karen, Nikos Aréchiga, and Marco Pavone. 2023. "Backpropagation Through Signal Temporal Logic Specifications: Infusing Logical Structure into Gradient-Based Methods." *The International Journal of Robotics Research* 42 (6): 356–70. https://doi.org/10.1177/02783649221082115.

Liang, Jacky, Wenlong Huang, Fei Xia, et al. 2023. "Code as Policies: Language Model Programs for Embodied Control." *IEEE International Conference on Robotics and Automation*.

Lin, Fanqi, Yingdong Hu, Pingyue Sheng, Chuan Wen, Jiacheng You, and Yang Gao. 2024. *Data Scaling Laws in Imitation Learning for Robotic Manipulation*. arXiv preprint arXiv:2410.18647. https://arxiv.org/abs/2410.18647.

Liu, Wenliang, Shuo Liu, Wei Xiao, and Calin A. Belta. 2025. *Feasibility-Aware Learning of Robust Temporal Logic Controllers Using BarrierNet*. arXiv preprint arXiv:2512.06973. https://arxiv.org/abs/2512.06973.

Liu, Wenliang, Nathalie Majcherczyk, and Federico Pecora. 2025. "Scalable Multi-Robot Task Allocation and Coordination Under Signal Temporal Logic Specifications." *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 3933–39. https://doi.org/10.1109/ICRA55743.2025.11128245.

Luo, Licheng, Kaier Liang, Cristian-Ioan Vasile, and Mingyu Cai. 2026. *Differentiable SpaTiaL: Symbolic Learning and Reasoning with Geometric Temporal Logic for Manipulation Tasks*. arXiv preprint arXiv:2604.02643. https://arxiv.org/abs/2604.02643.

Macenski, Steven, Francesco Soragna, Tully Foote Carroll, and Zoe Ge. 2023. "Impact of ROS 2 Node Composition in Robotic Systems." *IEEE Robotics and Automation Letters* 8 (7): 3996–4003. https://doi.org/10.1109/LRA.2023.3279614.

Mayne, David Q. 2014. "Model Predictive Control: Recent Developments and Future Promise." *Automatica* 50 (12): 2967–86.

Micheli, Vincent, Eloi Alonso, and François Fleuret. 2023. "Transformers Are Sample-Efficient World Models." *International Conference on Learning Representations*.

NVIDIA. 2026a. *NVIDIA Isaac ROS*. Https://developer.nvidia.com/isaac/ros.

NVIDIA. 2026b. *NVIDIA Isaac Transport for ROS (NITROS)*. Https://nvidia-isaac-ros.github.io/concepts/nitros/index.html.

Open X-Embodiment Collaboration. 2023. *Open X-Embodiment: Robotic Learning Datasets and RT-X Models*. arXiv preprint arXiv:2310.08864. https://arxiv.org/abs/2310.08864.

Park, Joon Sung, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. "Generative Agents: Interactive Simulacra of Human Behavior." *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*.

Pineda, Luis, Taosha Fan, Mark Monge, et al. 2022. "Theseus: A Library for Differentiable Nonlinear Optimization." *Advances in Neural Information Processing Systems*.

Pope, Reiner, Sholto Douglas, Aakanksha Chowdhery, et al. 2022. *Efficiently Scaling Transformer Inference*. arXiv preprint arXiv:2211.05102. https://arxiv.org/abs/2211.05102.

PyTorch Contributors. 2026. *Torchvision ResNet-18 Model Documentation*. Https://docs.pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html.

Raman, Vasumathi, Alexandre Donzé, Mehdi Maasoumy, Richard M. Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2014. "Model Predictive Control with Signal Temporal Logic Specifications." *IEEE Conference on Decision and Control*, 81–87.

Real-Stanford Diffusion Policy Contributors. 2023. *Diffusion Policy Image Workspace Configuration*. Https://raw.githubusercontent.com/real-stanford/diffusion_policy/main/diffusion_policy/config/train_diffusion_unet_image_workspace.yaml.

Seto, Danbing, Bruce H. Krogh, Lui Sha, and Alongkrit Chutinan. 1998. "The Simplex Architecture for Safe Online Control System Upgrades." *American Control Conference*, 3504–8.

SEU-PAISys. 2026. *Embodied.cpp Source Repository*. Https://github.com/SEU-PAISys/Embodied.cpp.

Silver, David, Thomas Hubert, Julian Schrittwieser, et al. 2018. "A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go Through Self-Play." *Science* 362 (6419): 1140–44.

Singh, Ishika, Valts Blukis, Arsalan Mousavian, et al. 2023. "ProgPrompt: Generating Situated Robot Task Plans Using Large Language Models." *IEEE International Conference on Robotics and Automation*.

Song, Chan Hee, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 2023. "LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models." *IEEE/CVF International Conference on Computer Vision*.

Song, Yang, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2021. "Score-Based Generative Modeling Through Stochastic Differential Equations." *International Conference on Learning Representations*.

Tedrake, Russ, and the Drake Development Team. 2019. *Drake: Model-Based Design and Verification for Robotics*. Https://drake.mit.edu.

Topcuoglu, Haluk, Salim Hariri, and Min-You Wu. 2002. "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing." *IEEE Transactions on Parallel and Distributed Systems* 13 (3): 260–74.

Wang, Guanzhi, Yuqi Xie, Yunfan Jiang, et al. 2023. *Voyager: An Open-Ended Embodied Agent with Large Language Models*. arXiv preprint arXiv:2305.16291. https://arxiv.org/abs/2305.16291.

Wu, Qingyun, Gagan Bansal, Jieyu Zhang, et al. 2023. *AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation*. arXiv preprint arXiv:2308.08155. https://arxiv.org/abs/2308.08155.

Xu, Ling, Chuyu Han, Borui Li, et al. 2026. *Embodied.cpp: A Portable Inference Runtime of Embodied AI Models on Heterogeneous Robots*. arXiv preprint arXiv:2607.02501. https://arxiv.org/abs/2607.02501.

Yao, Shunyu, Dian Yu, Jeffrey Zhao, et al. 2023. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models." *Advances in Neural Information Processing Systems*.

Yao, Shunyu, Jeffrey Zhao, Dian Yu, et al. 2023. "ReAct: Synergizing Reasoning and Acting in Language Models." *International Conference on Learning Representations*.

Yu, Gyeong-In, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. "Orca: A Distributed Serving System for Transformer-Based Generative Models." *16th USENIX Symposium on Operating Systems Design and Implementation*, 521–38.

Yue, Yang, Yulin Wang, Bingyi Kang, et al. 2024. *DeeR-VLA: Dynamic Inference of Multimodal Large Language Models for Efficient Robot Execution*. arXiv preprint arXiv:2411.02359. https://arxiv.org/abs/2411.02359.

Zhu, Yancheng, Ashok Kumar, Hugo Tadashi Kussaba, Wenliang Liu, Fares Abu-Dakka, and Abdalla Swikir. 2026. *Formal Methods for Robotic Manipulation: A Survey*. HAL Open Science.

Zidan, Arif Hassan, Yi Pan, Hanqi Jiang, et al. 2026. *World Models: A Comprehensive Survey of Architectures, Methodologies, Reasoning Paradigms, and Applications*. arXiv preprint arXiv:2606.00133.

Zou, An, Yuankai Xu, Yinchen Ni, et al. 2025. *A Survey of Real-Time Scheduling on Accelerator-Based Heterogeneous Architecture for Time Critical Applications*. arXiv preprint arXiv:2505.11970. https://arxiv.org/abs/2505.11970.