Where Does a Robot Think – On-Device vs Datacenter Inference

The Embodiment Problem, Planning vs Action Layers, Glass-To-Glass Budgets, Wafers & DRAM Constraints, One B300 vs 56 Thors TCO, Factories To Caves

IVAN CHIAM, ZANE FONG, BRYAN SHAN, AND 4 OTHERS

SEP 10, 2026 • PAID

1 1 Share

For most of its short history, AI lived behind a screen. That's starting to change.

First came chatbots, good for answering a question or drafting an email. Then agentic AI, models that don't just respond but do real work on a computer: navigating software, calling tools, finishing multi-step tasks on their own. Now the frontier is physical AI, intelligence that reaches past the screen to perceive the world and act on it. The biggest piece is robots, and it is still early: the hardware, the models, and the economics are all being worked out right now.

The Embodiment Problem

With LLMs, the hardware bends to the model. For hyperscalers and AI labs, there's no hard latency requirement and a very high spending ceiling, so the process is simple: pour in as much data and compute as possible at training, then figure out how to serve the model that comes out. The model comes first; the hardware serves it. Robotics inverts this. Tighter constraints flip the order of design: you first work out what it takes to serve a model on the robot, then build a model that fits. Two constraints drive this.

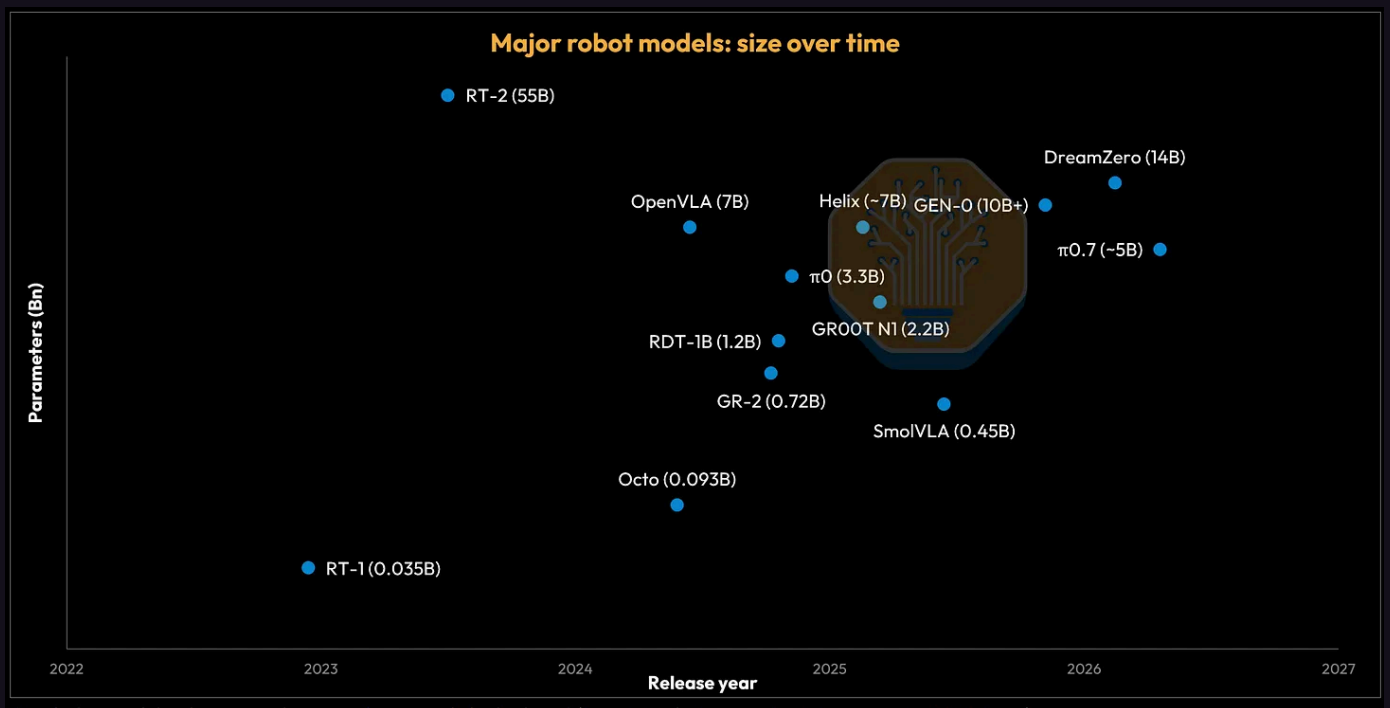
The first is time. A robot runs real-time control loops and can't miss a deadline. An LLM can be slow without affecting the final output. If a robot is slow, the world changes around it, and then the action is obsolete by the time it needs to enact it.

The second is cost. When you use an LLM, it lives behind a screen provided by the user. In robotics, the manufacturer has to build both the compute and the robot itself and pay for it upfront, on every unit, and the capital expenditure can be immense. At scale, this upfront cost can be billions and even in the trillions if we do eventually get to a billion robots.

into the robots and factories themselves, not into compute. Model capability is therefore capped by what can run on affordable, real-time hardware on the robot, and intelligence becomes something you ration against latency and unit economics.

Robot Models Are Still Small...

This is why frontier robot models remain far smaller than frontier LLMs, which have already pushed into the trillions of parameters. Today most frontier robot models sit between roughly 5 and 14 billion parameters: Physical Intelligence's π0.7 at 5 billion, and NVIDIA's DreamZero at 14 billion.



Source: SemiAnalysis

In LLMs, scaling laws have held, scaling has consistently broadened the range of problems a single model can solve. Early results show robotics might work the same way. Data has been a constraint but that is now scaling with World Models and Embodied human data collection. Embodied AI faces limits language models never did, like scarce data and the cost of real-world interaction, so whether scaling pays off the same way is genuinely an open question.

If robot models do continue to scale, the question becomes where should all that compute run, given the latency, cost and model limits mentioned above. The field is genuinely split,

its policy on-robot on a single H100. NVIDIA's DreamZero, a 14-billion-parameter "world action model" built on a video-diffusion backbone, needs two GB200 GPUs off-robot just to run in real time.

Our view is that a cascade is inevitable. Some robots will run cognition on-board, while others will offload it to GPUs sitting in a datacenter. The biggest robot model builders today, like NVIDIA and Google DeepMind, will lean toward this cloud approach, running the heavy cognition on cloud GPUs, with only a small edge model on the robot for low-level control and safety.

Smaller builders, such as Dyna and Sunday, may initially favor models that fit on a Jetson or consumer GPUs because their early deployments operate within narrow, controllable environments where onboard inference is sufficient. As deployments scale, however, they will converge on whichever mix of onboard and off-robot compute delivers the best performance and economics for their business model.

Moving the GPU off the robot brings real benefits, but also real challenges. Chief among them is latency, and it doesn't work everywhere. In environments with poor connectivity, cloud deployment isn't an option at all. Despite these difficulties, off-robot compute remains beneficial, and in some cases necessary, as the following sections show.

Models in Robots

Before we get into why the robot model should run in an off-robot datacenter, here's a quick primer on how robot models work, and which ones we're talking about.

Robot control is a broad space. Hard-coded control, where engineers write out the rules and math for every movement, has run factories for decades on embedded compute, and small task-specific policies fit onboard just fine. That's not our subject. We're talking about data-driven generalist robotics, robots run by large foundation models, because that's where the frontier is.

Even within data-driven generalist robot models, architectures vary. Some are monolithic: one model that goes straight from what the camera sees to how the motors move. Many are hierarchical: a big, slow model decides what to do, and a small, fast one turns that into movement, hundreds of times a second. Our argument for off-robot datacenter compute is

The planning layer is the slow, high-level one: it perceives the scene, reasons about space, and plans the task. The action layer is the fast, low-level one: it turns those plans into motor commands. The planning layer, which is also the heavy part of the model, is the more forgiving: it updates only a few times a second and runs asynchronously from the control loop, so the action layer keeps acting on the most recent plan while the next one is still in flight. That decoupling lets planning absorb the delay, and even the jitter, of a wireless round trip, while the action layer runs at hundreds of Hz and can tolerate none of it. So planning is the natural candidate to move into the datacenter, while the fast control loop stays local.

Source: SemiAnalysis

The amount of planning scales with how general the robot's job is. A generalist built to handle open-ended tasks needs a large planning model doing genuine reasoning, and those models are heavy. That reasoning is exactly what gets pushed to the datacenter. A specialist is the opposite case: a narrow policy doing one repetitive job carries little high-level planning, so the whole model is small enough to stay on the robot.

Why DC GPUs are better suited...

Specialist robots are for the most part solved: factories have run on them for many years now using hard-coded automation, but not modern AI. But a specialist is rigid, one task at

product mix, and the long tail of unstructured work. This aligns with some of the many lessons of the Toyota Production System, where we learned that multi-skilled workers can reduce waste and improve economic efficiency. As a result, the TAM for generalist robotics is huge, not just in manufacturing and warehousing but also in homes and other places.

The more generality you demand, the more you lean toward FLOP-hungry foundation models and datacenter-scale compute. Not every robot needs maximal generality, and many tasks are served well by smaller models, but the frontier of flexible, general-purpose robots is where datacenter-scale compute becomes necessary.

LLMs vs Robot Models Token Flow

An LLM ingests and emits huge, bursty token counts: a long prompt to prefill, then a long generation, with a KV cache that grows the whole time. That is the memory wall, the capacity to hold that cache and the bandwidth to stream the weights and cache on every token, and it makes LLM serving bandwidth-bound.

A robot model is the opposite, a smaller steady stream, more metronome-like rather than a burst. It never stops, because a robot is not answering one question and halting; it is locked in a closed loop with a moving world, taking in a fresh frame and emitting a short action chunk a few times a second. Each frame is compressed in dedicated hardware (the ISP and video-codec ASICs) into a fixed, compact token budget of tens to a few hundred tokens. So robot models are not as memory capacity and bandwidth bound in the same way that LLMs are.

Source: SemiAnalysis

Frontier Robot Models are FLOPs hungry

What binds a frontier robot model sometimes memory, but often it is the FLOPs to run it.

These are FLOP-hungry workloads: a VLA (Vision Language Action model) is a multi-billion-parameter model, orders of magnitude larger than the specialist policies before it, and its vision and language stages, which are the bulk of the work, are compute-bound on any capable GPU. A WAM (World Action Model) is heavier still. Instead of naming an action, it generates video of the predicted future and derives the action from it, through iterative denoising, meaning one action could take multiple full forward passes through a model.

Edge silicon cannot keep up. Nvidia's latest Jetson Thor, the best-in-class robot brain, has only about a tenth of a single B200's compute, and roughly a fourteenth of a B300's. DreamZero (a 14B WAM) needed two GB200 GPUs just to reach ~7Hz, which is on the order of twenty times the compute a single Thor can muster. Run the same model on a Thor and it would fall to a fraction of 1Hz, far too slow for real-time control.

Source: SemiAnalysis

Source: SemiAnalysis

Shipping the Robot Model Off Device

If a Jetson Thor can't keep up, and we off-load the heavy, compute-intensive part of the workload to datacenters, it means optimizing local compute for safety loops and high-frequency control loops, and doubling-down on network interfacing.

the most when transmitting image data.

Transmitting image data quickly and reliably is more than just a dedicated compression ASIC on-device and calling it a day; the entire upstream image processing pipeline must be optimized. Noise is the enemy of compression, the less spatiotemporal structure an image has, the harder it is to compress. So a good local compute system optimized for networking will prioritize the components that keep noise out of the image in the first place: optics, low-light sensor performance, sensor SnR, thermal management, and the image signal processor (ISP).

ISPs turn messy analog signals into digital ones, but can ruin compression efficiency when not optimized for compression directly. And when we consider the imaging pipeline as a whole, we are optimizing for the model, not for human viewing. This is similar to how WAMs do not require the extra denoising steps that typical video diffusion models use; they solve a different task. The same principle applies here: we care about good model outputs and a low bitrate, not necessarily beautiful images (see below).

Left image is more pleasing to the eye, but the right image has a higher SnR.
Source: NASA

Network conditions introduce a lot of variability, so local compute must also incorporate dynamic, real-time adaptation of both the ISP and the compression bitrate, similar to

A pipeline that produces a clean, adaptable stream also needs hardware explicitly tuned for offloading – getting the bits off-device and on-schedule. Historically, mobile chipsets prioritized downlink speeds, but robotics requires massive uplink capacity to stream out high-fidelity telemetry. The download can be relatively small. This means multiple transmit antennas with proper placement, since the environment and robot itself can block signals. It also means support for a wider array of RF bands to allow multiple channels for higher throughput and less congestion – especially with factory conditions in mind. Optimizing hardware and embedded firmware for fast handovers between access points and between WiFi and 5G is critical since the typical handover time will temporarily kill the latency budget. Jitter is the biggest enemy.

The end goal is a highly coordinated hybrid architecture that maintains a healthy, but not overly indulgent, amount of FLOPs locally for safety and high-frequency control and can reliably off-load the "foundation" part of a foundation model. It's not a perfect system: simple control loop fallbacks are still required the moment the network connection diminishes or is completely lost. We require latency watchdogs, heartbeat timeouts, and staleness bounds on remote actions.

Power

Datacenter GPUs have to run off the robot, because the battery can't feed them. A single Blackwell GPU draws ~1.2-1.4 kW and has to be liquid-cooled, built to sit in a rack. A humanoid, by contrast, stores only about 2 kWh in its battery and draws a few hundred watts in normal motion, a kilowatt or two at peak, everything included, which is why its onboard brain is a 40 to 130 W Jetson Thor. Put a datacenter GPU on the robot and the chip alone would out-draw the whole machine, demand cooling it can't carry, and flatten the battery in minutes. Offloading sidesteps the power constraint. The robot carries only a 20 to 30 W perception compute + wireless board to reach the datacenter GPU, comfortably within budget.

The Supply-Chain Wall

Nvidia's accelerator output is almost entirely datacenter silicon: a long Hopper ramp through 2023 and 2024 handing off to Blackwell, which scales sharply across 2025 and 2026 to dominate the mix. Jetson, the line used in robots, is the thin red sliver along the bottom,

Source: SemiAnalysis Accelerator Model, we withheld Rubin shipments on this chart, buy the data

Margins tell a similar story. Demand isn't there yet, but when it arrives, pricing will go up and Jetson margins will inflect. But for now, a Blackwell datacenter GPU earns far higher gross margins than a Jetson module, on the order of mid-to-high 70s percent versus mid-60s, so NVIDIA has every incentive to point its scarce leading-edge wafer capacity at the datacenter and little reason to ramp edge silicon while the robotics market stays small. Robotics chips like Jetson Thor also require a lot more memory per unit of compute which is a major roadblock.

Source: SemiAnalysis Accelerator Model

Worse, robot silicon is converging onto the exact nodes the datacenter is fighting over. Nvidia's Jetson line used to be insulated: the previous generation, Orin, ran on Samsung's SF8 process. The current generation, Thor, has closed that gap onto TSMC N4, which Blackwell uses, and as robots run on more capable models their silicon has to stay on the leading edge: the generation after Thor likely moves to N3 alongside Rubin, then N2 alongside Feynman, exactly the nodes the entire AI-accelerator roadmap is fighting over. So a low-volume, lower-margin Jetson will always be competing for advanced-node wafers from the back of the queue.

Source: SemiAnalysis Accelerator Model

Source: SemiAnalysis Accelerator Model

But the wafer demand from robots is not where the strain is. A million Jetson-class chips in 2030, each around 400mm^2, is only about ten thousand wafers for the year, a rounding error next to the datacenter GPU ramp, because Thor's die is barely half a reticle size chip, and there simply aren't many robots yet. Supply only tightens once robots need tens of millions of chips a year, well beyond 2030. So the real question is not whether robots can get wafers. It is silicon efficiency: for a given fleet of robots, which approach consumes less leading-edge silicon? Putting a chip in every robot, or serving their cognition from shared datacenter GPUs.

Source: SemiAnalysis Accelerator Model

If you look at silicon efficiency as wafers per robot, the onboard chip and the shared datacenter GPU cross over at around 7 robots per GPU. Beyond that, serving cognition from a shared GPU takes less silicon per robot than putting a chip in every machine. And that's what really matters here, because when fab supply is scarce, the approach that uses less silicon per robot is the one that scales.

Source: SemiAnalysis

Another major bottleneck is memory. Each Jetson generation has carried more DRAM than the last: Jetson Xavier used 32GB, AGX Orin 64GB, and Jetson Thor now uses 128GB of LPDDR5X, with future generations set to keep climbing. Although total DRAM wafer capacity continues to grow, most of the incremental capacity is being absorbed by HBM for AI accelerators.

That leaves the commodity and LPDDR supply that robot brains depend on competing for a shrinking pool of non-HBM wafers. Similarly, when DRAM is the scarce resource, the approach that uses less DRAM per robot is the one that scales. The LPDDR is better off being allocated to Vera GPUs or wafers to HBM.

Source: SemiAnalysis Memory Model

Memory tells a similar story, the onboard chip and the shared datacenter GPU cross over at around 5 robots per GPU. Beyond that, serving cognition from a shared GPU takes less DRAM per robot than putting a chip in every machine.

Source: SemiAnalysis

The TCO Verdict: One B300 server vs. 56 Thors

We ran DreamZero, current leader on RoboArena, on a real B300, using the inference optimizations from NVIDIA's WAM paper (CUDA graphs, DiT caching, NVFP4 quantization, moving the scheduler onto the GPU). We also ran a single denoising step to match the DreamZero-Flash setup, though without Flash's retraining recipe, so task quality is unvalidated.

DreamZero's serving profile is prefill-shaped: each inference pushes a full chunk of video latents (thousands of tokens) through the 14B DiT in parallel, so a B300 is compute-bound at batch size 1. Batching therefore buys nothing: FLOPs scale one-for-one with batch size, and every robot just waits longer. Instead, we time-multiplex, with the B300 serving each robot sequentially. This is specific to video-generating WAMs. A VLA like $\pi 0$ or GR00T flips the roofline: a few hundred tokens through a small backbone leaves the bandwidth-bound, so batching amortizes the same weight reads across robot throughput scales nearly linearly with batch size. Serve a VLA fleet with con batching; serve a video WAM one robot at a time.

人在草木间

1.6 seconds our latency budget: the total time a request gets to spend across network, queueing, and compute before the robot runs out of instructions and stalls mid-motion.

Source: Nvidia

We read the chart below on p99 latency (blue line), the 99th percentile response time. 99 out of 100 requests come back faster than this, so it's effectively the worst case. It matters more than the median (green line) here because a robot stalls the moment any single chunk shows up late. We see that the maximum number of robots we can process on 1 B300 GPU is 7 robots. At 7 robots, p99 is 1.16 seconds, still under the wall, with zero missed chunks over the run.

Source: SemiAnalysis

Now that we know we can time-multiplex 7 robots per GPU, the fundamental TCO question stands:

Is it cheaper to offload compute to a B300 server, or to run inference on-device in a Jetson Thor?

We build up two scenarios: the offload scenario assumes one Blackwell-class server which can serve 7 robots (i.e. one B300 NVL8 server serving 56 robots), and an on-robot scenario of the same 56 robots each carrying its own Jetson Thor. In both scenarios, common parts across both cases (i.e. mechanical components, robot shells) are not included in the BOM cost calculation.

We also discuss the biggest challenge with this deployment mechanism.

This post is for paid subscribers

Subscribe



semianalysis

Previous

© 2026 Dylan Patel · Privacy · Terms · Collection notice

Start your Substack

Get the app

Substack is the home for great culture