A Brain Too Big to Carry — On-Device vs Datacenter Inference

Robot Models, Silicon & DRAM Efficiency, Jetson Thor vs. B300 TCO, Deployments, The Network Wall

IVAN CHIAM, GIANLUCA, ZANE FONG, AND 3 OTHERS

SEP 15, 2026 • PAID

34 1 1 Share

Where should the brain of the robot go?

So far, AI has mostly lived behind a screen. Chatbots answered questions. Then agents started driving software and finishing multi-step tasks on their own. The next step is AI that acts in the physical world, and the biggest piece of that is robots. It's early. Nobody has settled the hardware, the models, or the economics.

The Embodiment Problem

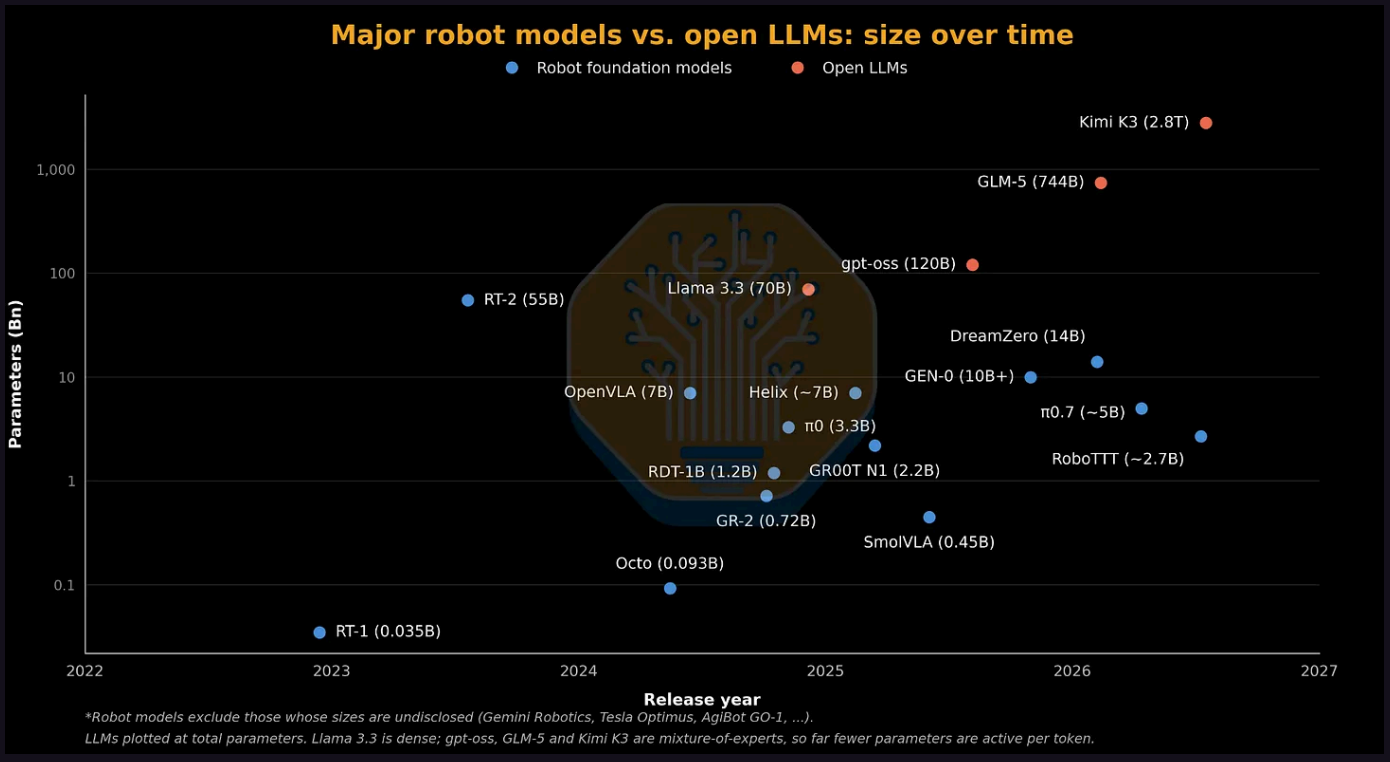
With LLMs, the hardware bends to the model. Pour in as much data and compute as possible at training, then figure out how to serve the model that comes out. The model comes first; the hardware second. Robotics inverts this due to two constraints.

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 If a robot is slow, the world changes around it, and then the action is obsolete by the time it needs to enact it.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, even in the trillions if we do eventually get to a billion robots.

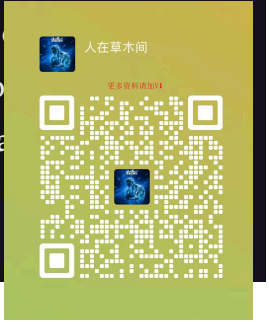
Because of these two constraints, in robotics the on-robot hardware is fixed, and the model is designed to fit what we can reasonably manufacture. Model capabilities at the frontier will be capped by what we can run on affordable, real-time hardware, and intelligence becomes something you ration against latency and unit economics.

already pushed into the trillions of parameters. Today's generalist robot models are in the billions of parameters: Physical Intelligence's π0 class at ~3 billion, π0.7 at 5 billion, ByteDance's GR-3 at 4 billion, Generalist's models around 10 billion, and NVIDIA's DreamZero at 14 billion. However, these parameter counts carry less meaning than their LLM equivalents. LLM labs pick model size by optimizing quality against a training budget and inference cost; robotics labs pick size by what their data can support and what fits on a Jetson or an H100 within a latency budget. Robot model sizes tell you where the constraints sit, not where sizes will settle.



Source: SemiAnalysis

In LLMs, scaling laws have held, scaling has consistently broadened the range of problems a single model can solve. Early results from companies such as Generalist and Dyna suggest robotics scales the same way. The question isn't whether scaling works, it's whether the inputs can keep up. Data is the first constraint, there is no internet-scale corpus of robot experience, and real-world interaction data is slow and expensive to generate hence the rise of embodied human data collection and world models. The network is the se
constraint. Frontier models already outgrow the robot (π0.7 runs on an off-ro
DreamZero needs two GB200s), and once the model lives in the datacenter, la
jitter pose problems.



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. But compute requirements change rapidly: months after DreamZero, NVIDIA's RoboTTT made the opposite bet, using a 3-billion-parameter policy that continually updates its own weights with test time training instead of generating videos of the future. The result is a model with minutes of usable context that is small enough to run onboard.

Wherever model sizes land, our view is that a cascade of approaches is inevitable. Some robots will run cognition fully onboard, while others will offload some part of it to GPUs sitting in a datacenter. However, for generalist robots that require real intelligence, we think off-robot compute has many advantages, from escaping the robot's compute and power budget to pooling inference across a fleet. It comes with network challenges, which we cover in later sections, but it is an increasingly viable option.

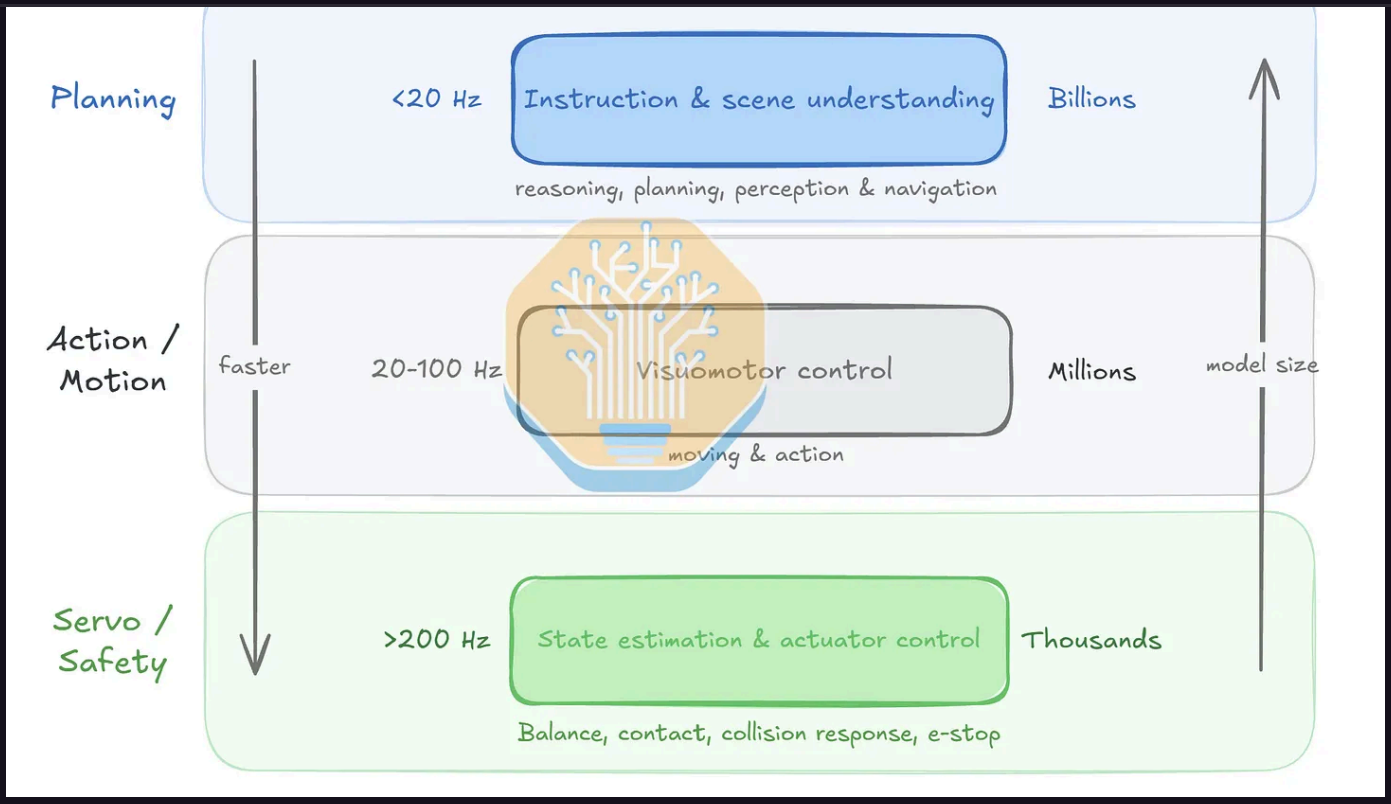
Primer on Robot Models

Before we explore the idea of off-robot compute, here's a quick primer on how robot models work, and which ones we're talking about.

Classical robot control, kinematics plus planned trajectories plus feedback controllers, has run factories for decades on small embedded chips. Plenty of narrow learned policies fit onboard too. Neither is what this piece is about. We care about generalist robots: ones that can follow open-ended instructions, handle scenes they've never seen, and do many different jobs.

Layers of a robot model

At the top is the planning layer. It interprets the instruction and the scene, decides what the robot should do next, and may produce a subtask, destination, grasp pose or footstep plan. Below it is the action or motion layer. It turns those outputs into short action sequences, body poses, or velocities. At the bottom are the servo and safety loops. They estimate the robot's state, react to contact and slip, maintain balance and ultimately command the actuators.



Source: SemiAnalysis

Frequency determines which layers need to run on the robot. The action and servo layers runs at hundreds of Hz: a 100 Hz loop must produce its next output every 10 milliseconds. A typical wireless round trip costs 10 to 50 milliseconds on its own, and a well-engineered link takes less than 10 milliseconds, so for these loops the network does not merely add delay, it consumes almost the entire budget before the model has computed anything. This is why anything above roughly 100 Hz can never leave the robot.

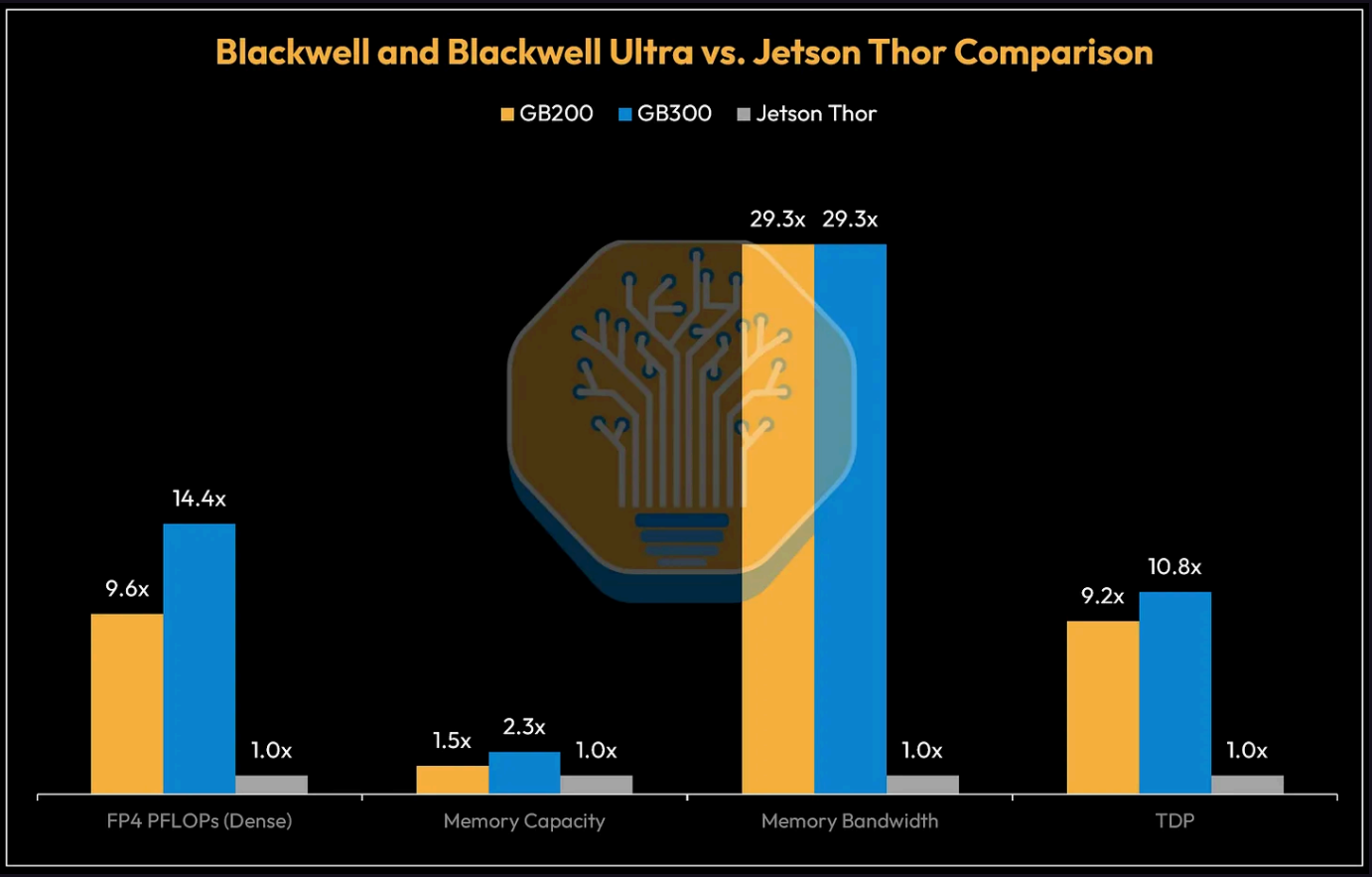
Below roughly 20 Hz, where the planning layer lives, the picture changes. A planner running at 5 Hz gets 200 milliseconds per decision, so even a 10 to 50 millisecond round trip fits comfortably, with margin to spare. That is enough headroom to move the planning layer off the robot entirely, and a well-tuned network stack could push the offloadable boundary above 20 Hz. The bigger problem, though, is jitter rather than latency. A fixed delay can be planned around: the system knows roughly how far the robot and everything around it will move in that window and can plan a little ahead. Jitter is harder because commands arrive at irregular intervals, so the robot never knows exactly when the next update will land. You can buffer to the worst case, which turns variable delay into predictable delay, but then every action pays the worst-case penalty. Getting jitter under control is what actually unlocks offloading the planning layer.

and reasoning models are big. That is why some companies have started looking at datacenter compute instead. Jetson Thor is the best robot compute you can buy today, and it still delivers only about 1/10th the FLOPs of a GB200 and roughly 1/30th of its memory bandwidth.

Datacenter vs Edge: NVIDIA Accelerator Specs			
	Blackwell		Jetson Series
Accelerator	B200/ GB200	B300/ GB300 (Ultra)	Jetson AGX Thor (T5000)
GPU TDP (W)	1,000/1,200	1,200/1,400	40-130
Foundry Node	4NP		4NP
FP4 PFLOPs - Dense (per Package)	9/10	14/15	1.04
FP8 PFLOPs - Dense (per Package)*	4.5/5	4.5/5	0.52
FP16 PFLOPs - Dense (per Package)*	2.3/2.5	2.3/2.5	0.26
HBM Capacity	192GB HBM3E	288GB HBM3E	128GB LPDDR5X
HBM Stacks	8		-
Memory Bandwidth	8TB/s		273 GB/s

*NVIDIA publishes only FP4 (2,070 TFLOPS sparse) for Jetson Thor; FP8 and FP16 dense are scaled from FP4

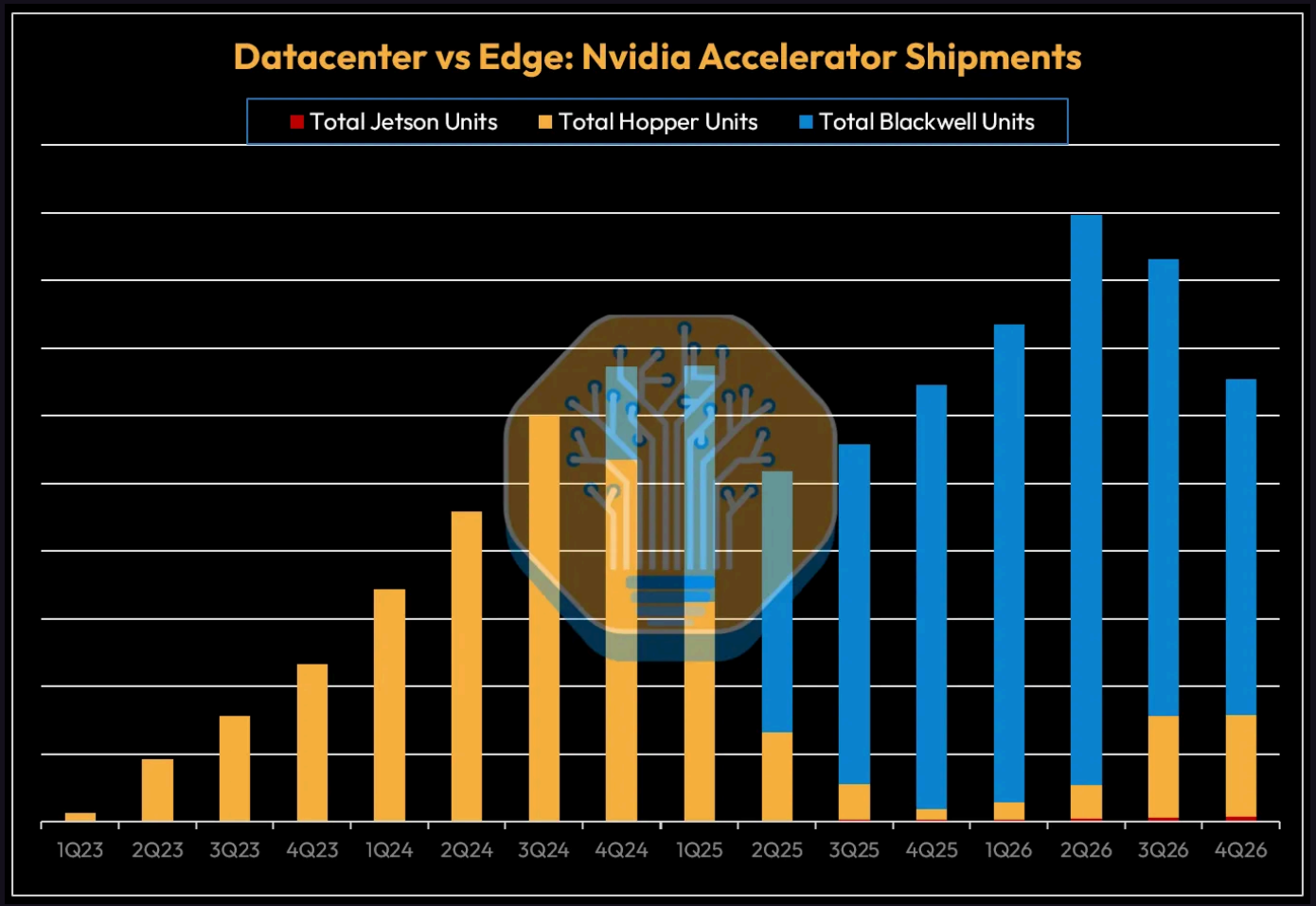
Source: SemiAnalysis



Source: SemiAnalysis

The supply chain is also geared towards datacenter silicon, not robot silicon, and ramping robot silicon will not be easy, especially with front-end capacity and DRAM already tight. Nvidia's own output makes the point.

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, barely visible even in 2026. This is because the robotics market is nascent, real volume is years out, and there is next to nothing to build for.



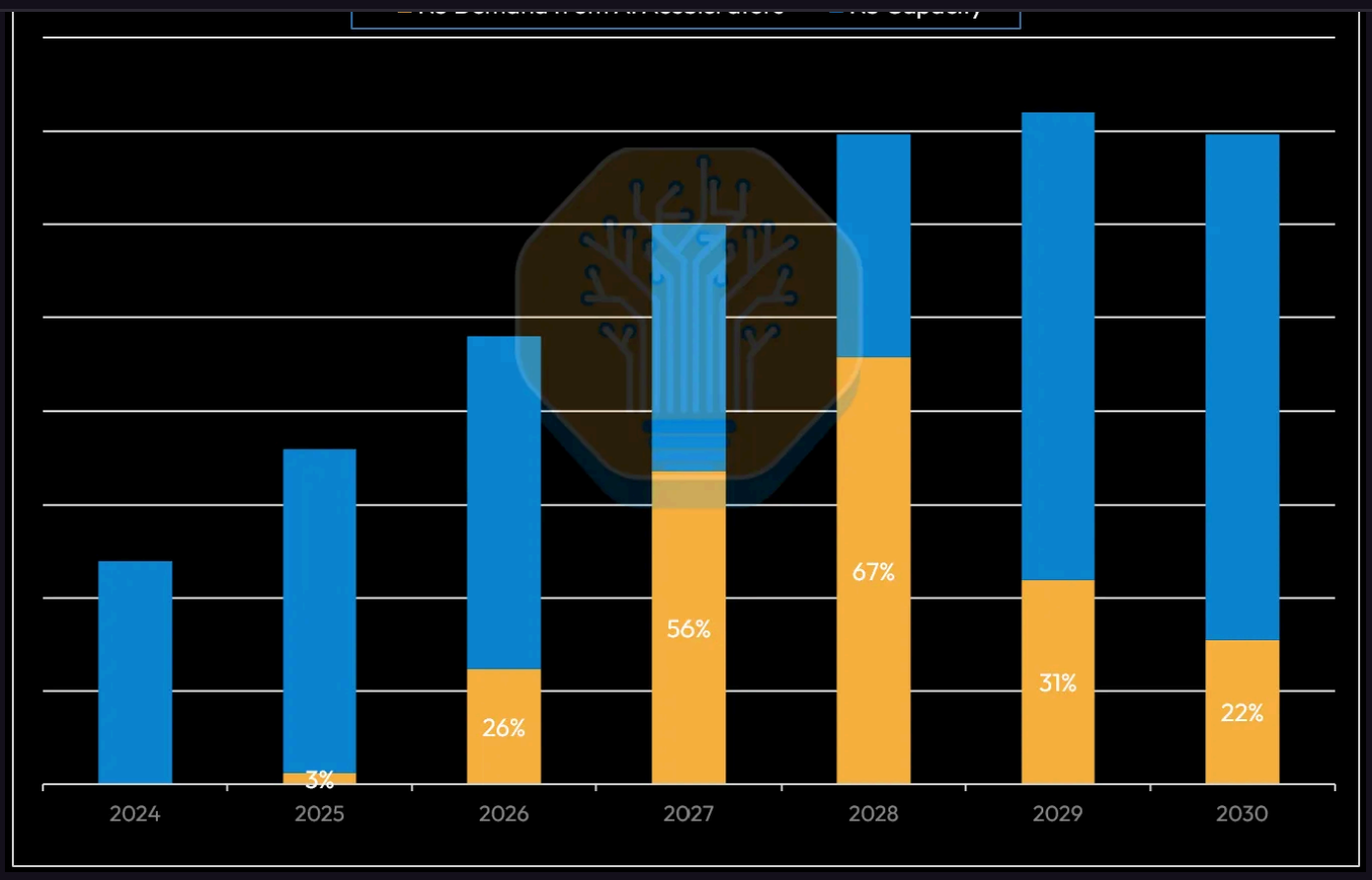
Source: SemiAnalysis Accelerator Model, note Rubin withheld from chart, ramp provided to clients

Margins tell a similar story. Demand isn't there yet, but when it arrives, pricing should 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.

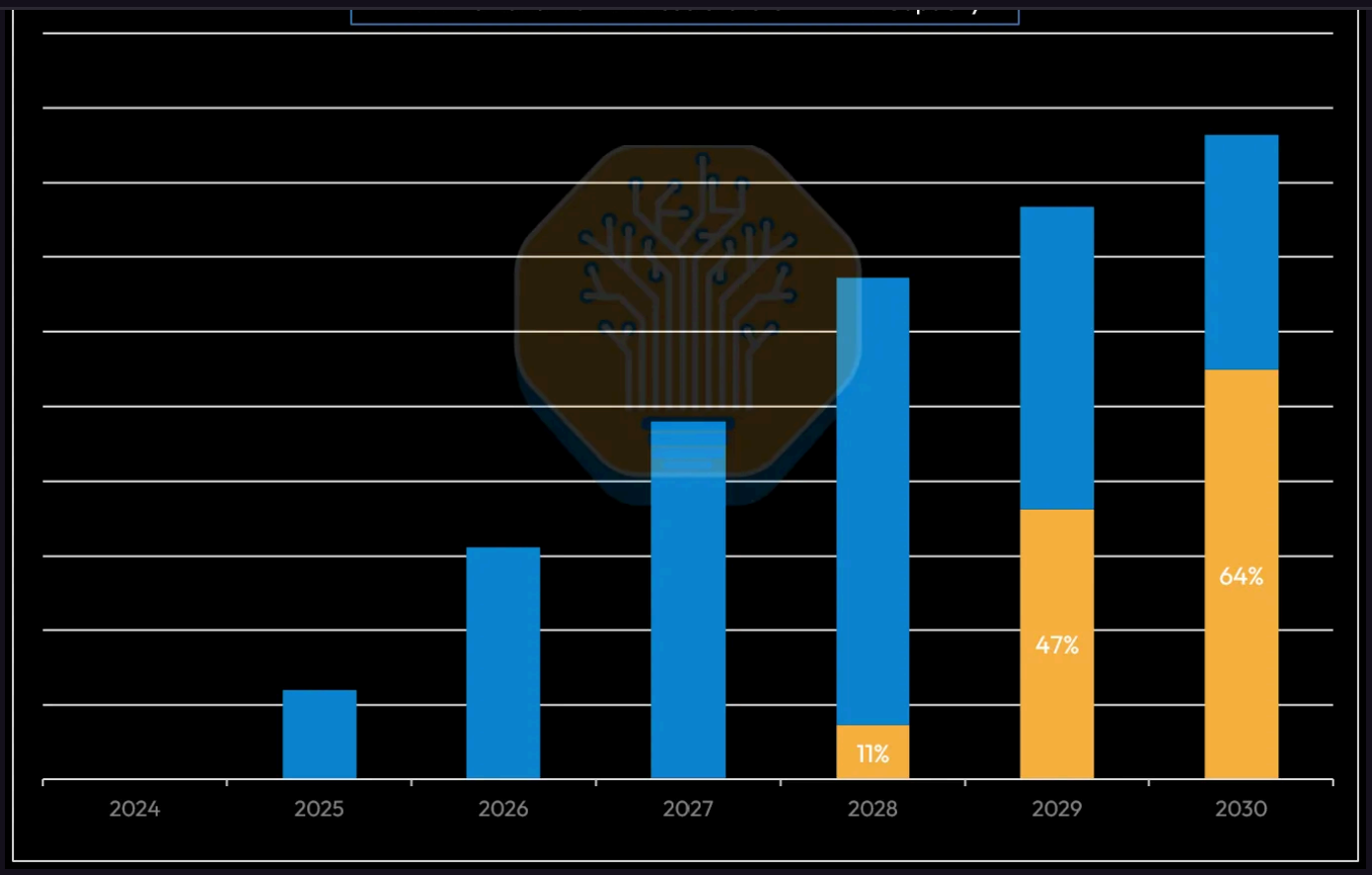
BOM & margin analysis	
Compute die cost	[redacted]
Substrate	
Assembly + test + packaging	
PCB	
VRM	
LPDDR5X	
Total BOM	**1,103**
Margin	63%
ASP	**3,000**

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, 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.

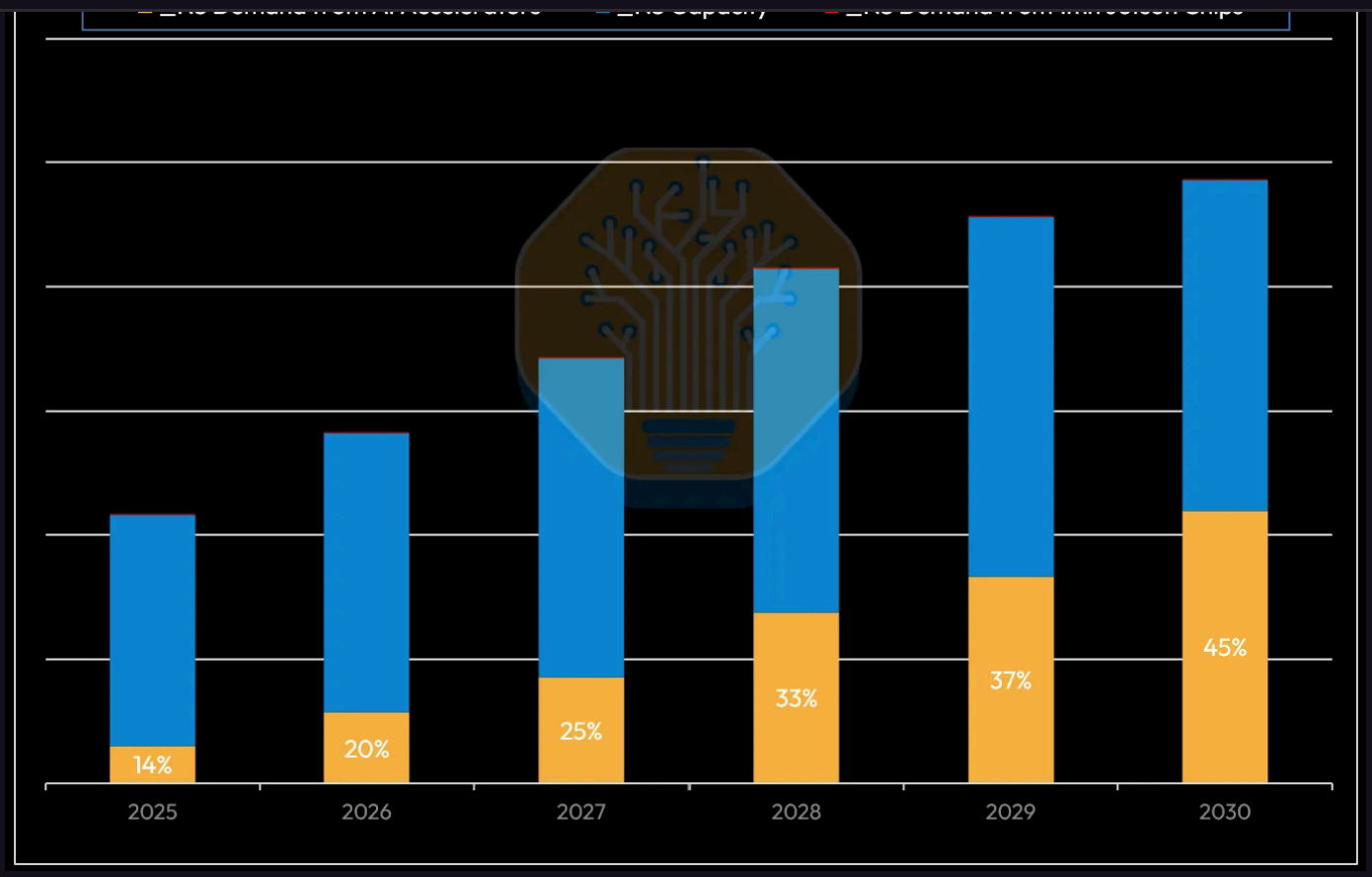


Source: SemiAnalysis Accelerator Model



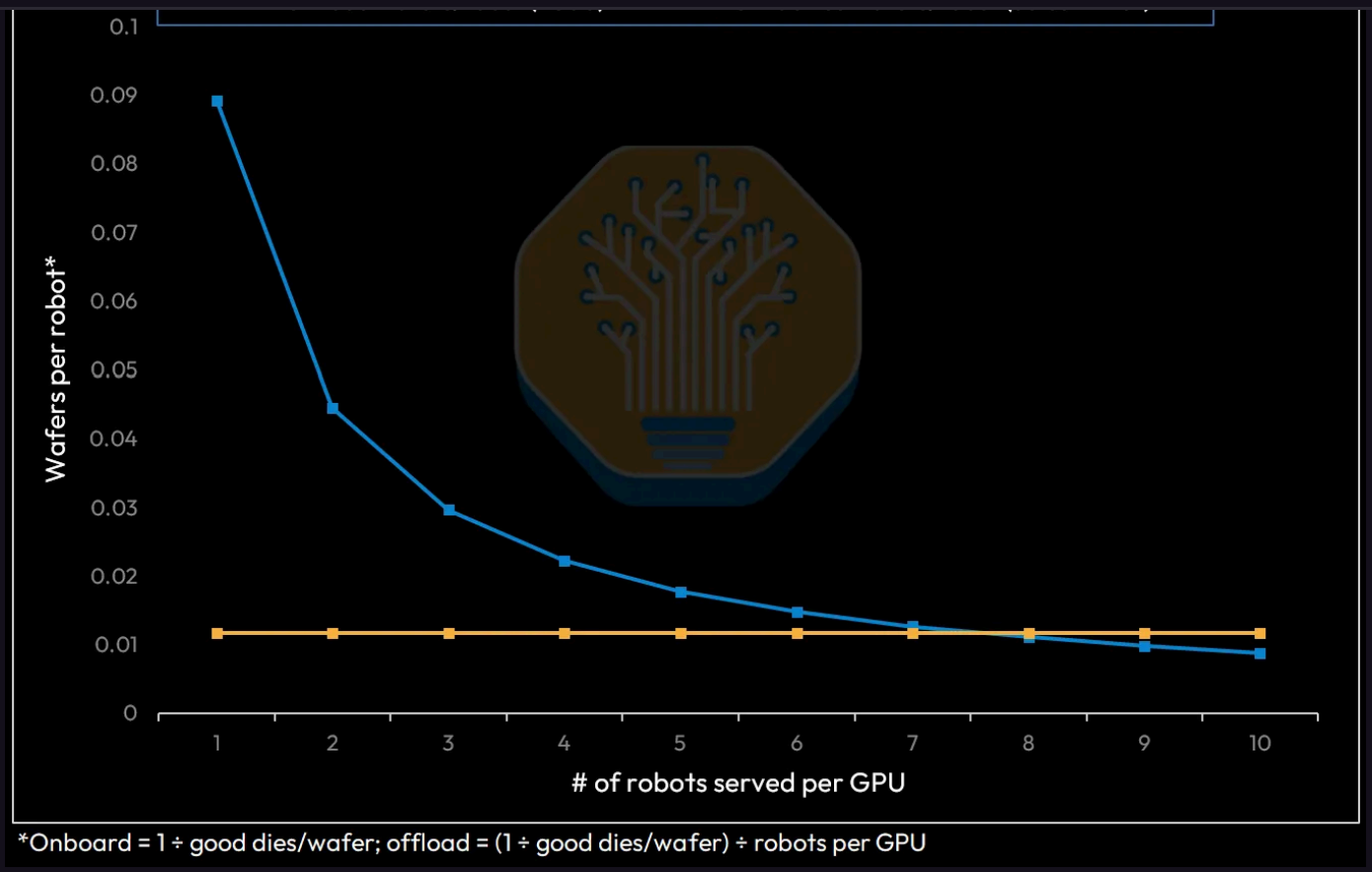
Source: SemiAnalysis Accelerator Model

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-sized 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.



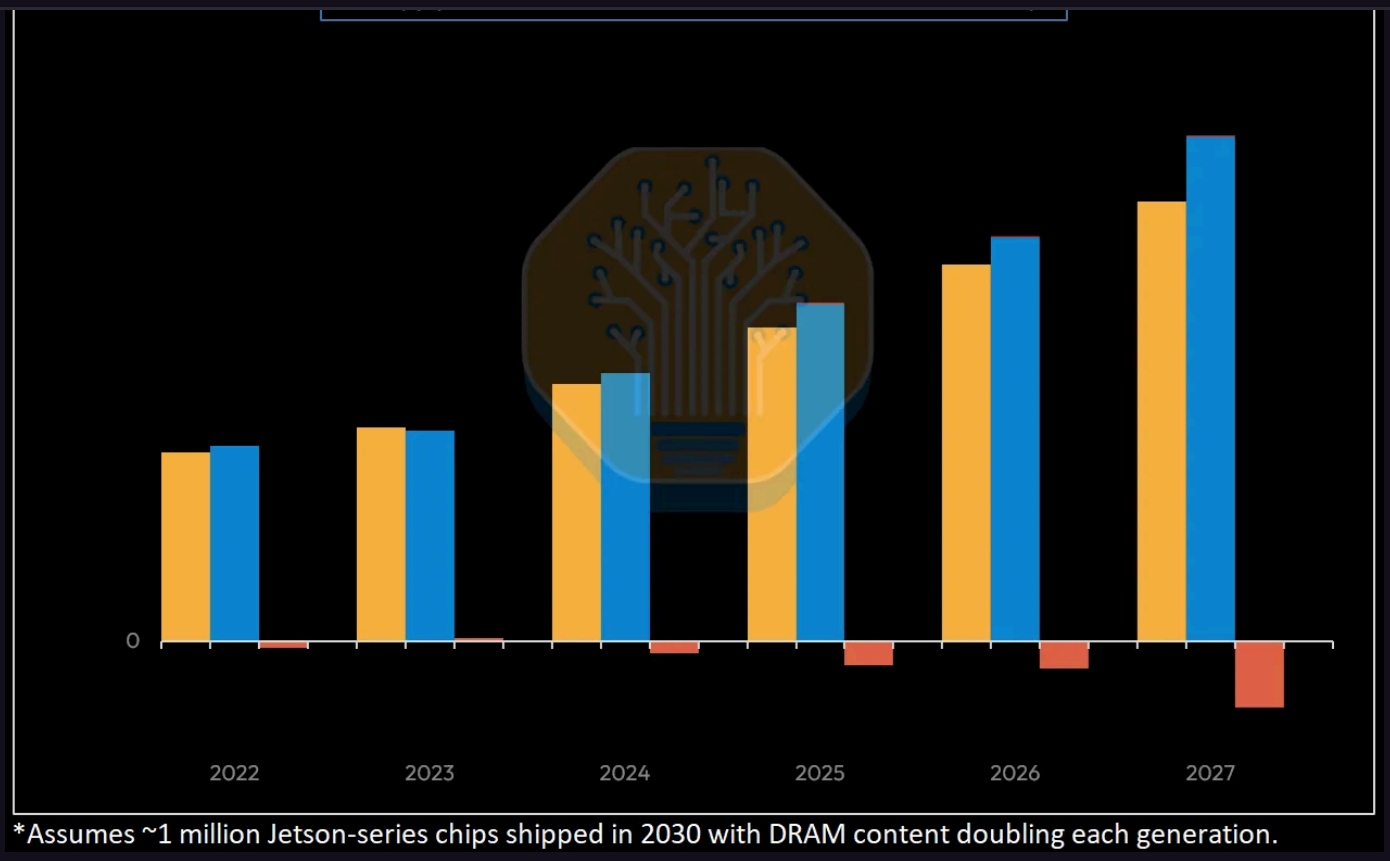
Source: SemiAnalysis Accelerator Model

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.



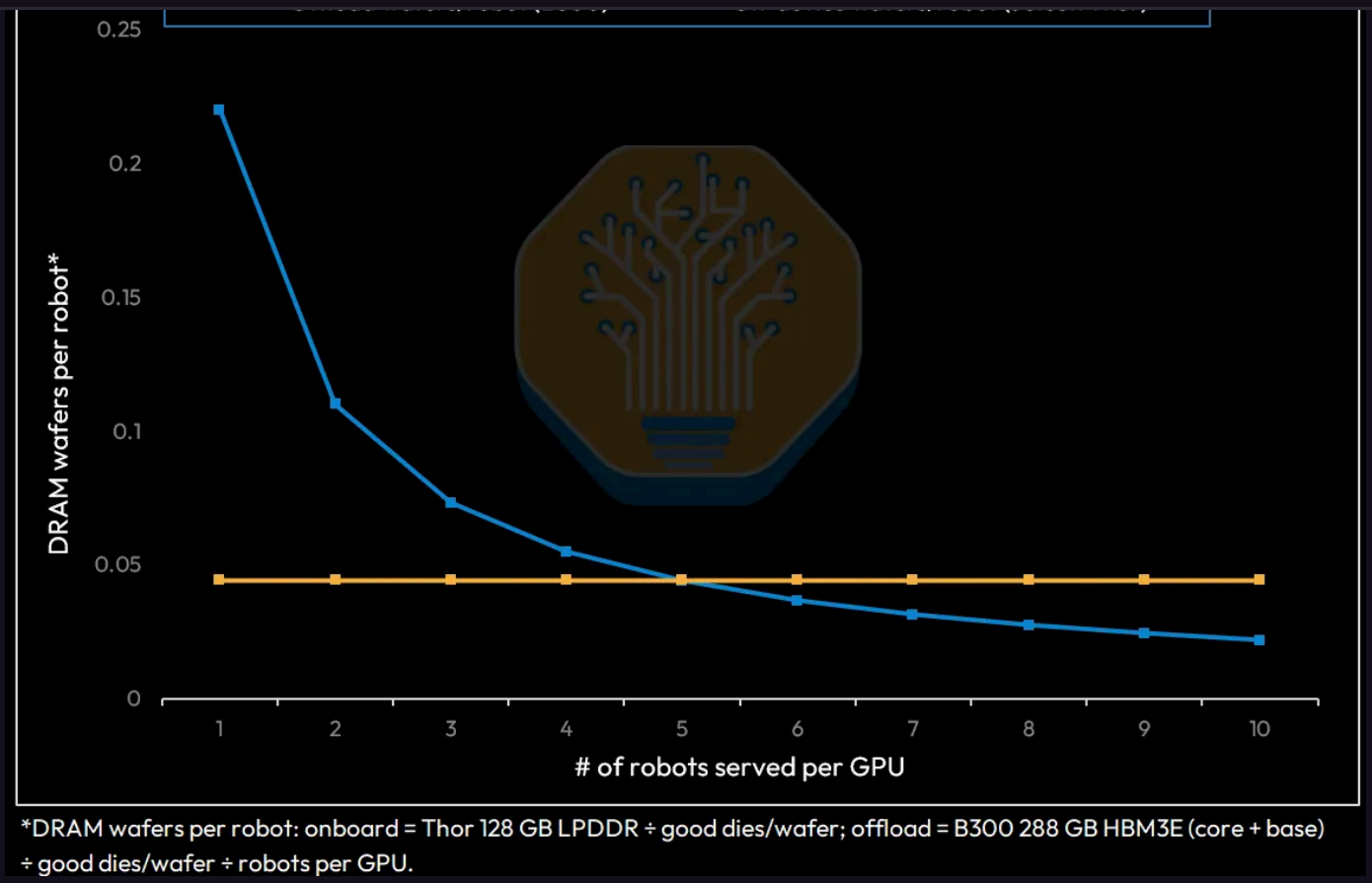
Source: SemiAnalysis

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.



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

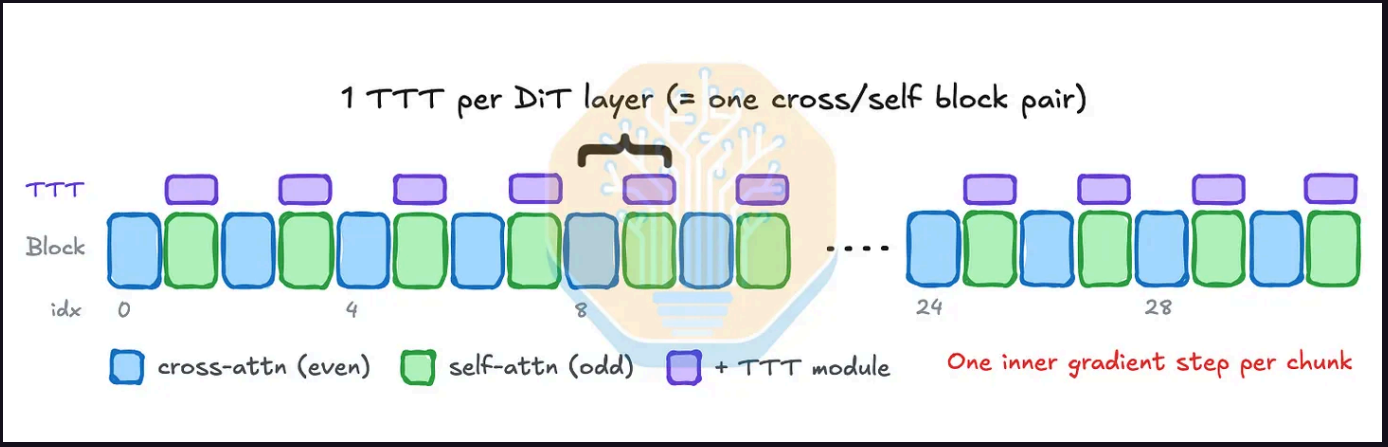
The TCO Verdict: B300 vs. RTX 6000 PRO vs. Jetson Thor

Wafers and DRAM tell us which approach can scale, but they say nothing about which one is cheaper to run. So the next question is cost. If you had to serve the same robot fleet, what would it take with B300 servers in a datacenter, RTX 6000 Pro servers in a datacenter, or a Jetson Thor in every robot?

We use NVIDIA's RoboTTT. It has no public code or weights, so we benchmark a timing-representative reconstruction: real GR00T N1.7 checkpoints with test-time-training (TTT) blocks inserted into the action head, built to match the paper's compute and memory profile rather than its accuracy.

Three details for faithfulness. GR00T's DiT interleaves cross-attention and self-attention blocks by index, so a naive fixed stride would place every TTT module on the same block type; we insert 16 modules as (even, odd) pairs, 8 following cross-attention and 8 following self-attention across the 32 blocks. We prepend 16 learnable register tokens, giving the

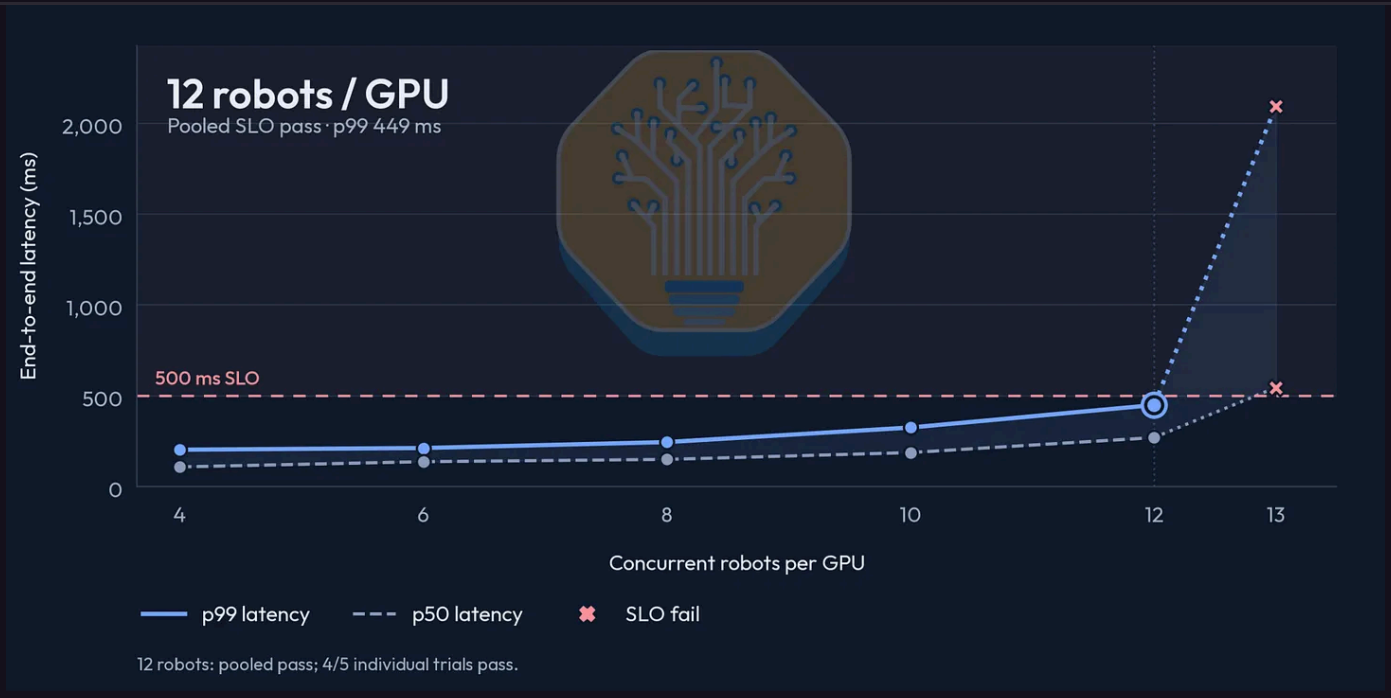
robot carries 151 MB of fast-weight state, gathered before every batched call and scattered back after, which is the multi-robot serving cost that a shared-weight VLA does not pay.



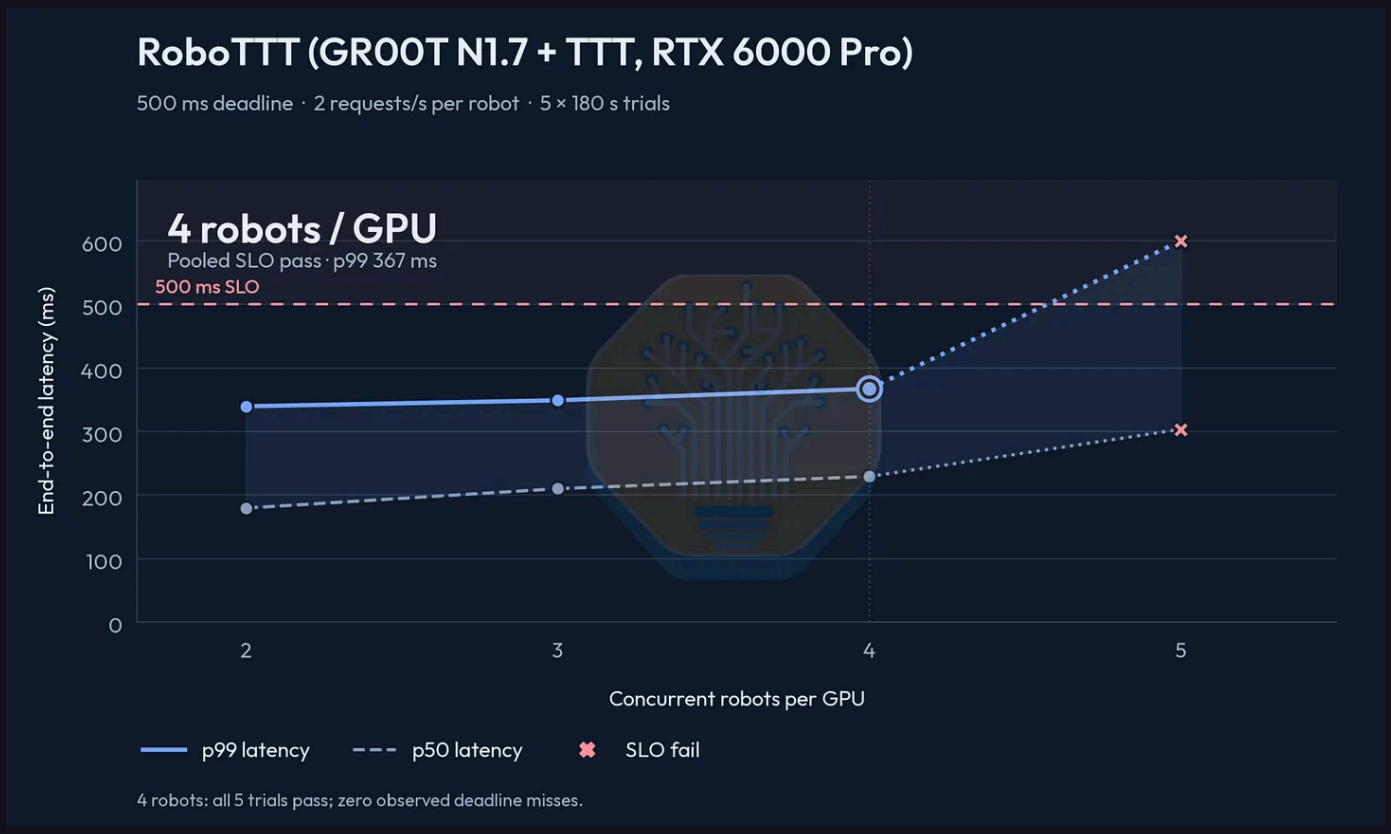
Source: SemiAnalysis

Thus, the TTT path costs full FLOPs, memory traffic, and state swapping while leaving the emitted actions untouched. Untrained fast weights diverge and produce garbage, and we are measuring serving cost instead of task success. These numbers say how expensive RoboTTT-shaped inference is to serve.

On that basis, one B300 sustains 12 robots within the 500 ms chunk deadline, against 4 on an RTX PRO 6000.



Source: SemiAnalysis



Source: SemiAnalysis

With this, we build up three scenarios. For the offload scenarios, we assume one Blackwell-class B300 server which can serve up to 12 robots per GPU, as well as an RTX 6000 Pro

In all three scenarios, common parts across both scenarios (i.e. mechanical components, robot shells) are not included in the BOM cost calculation.

B300 offload scenario

The first offload scenario is one B300 server, plus 96 wireless boards to connect the robot to the GPU server wirelessly (one per robot, twelve robots per GPU at the benchmarked p99 criterion). That brings all-in capex to ~$554K, which, including operating costs like colocation and power, leads us to an aggregate all-in TCO of $18.63/hr in the offload scenario. This assumes owner/operator economics, not a cloud rental quote.

Cost of Offloading Compute to GPU Server

		B300 1200W
Server Cost	USD	
Server Service	USD	
Networking Cost	USD	
Software Licenses and Other Costs	USD	
Other Installation	USD	
Total Cluster Capital Cost	USD	
Wireless TX/RX Card	USD	
Total Capital Cost	**USD**	**$554,434**
Useful Life	Years	6
WACC	%	10.3%
Total Capital Costs per Month	USD	$10,355
Total Operating Costs per Month	USD	$3,242
Capital Cost, per Hour	USD/hr	$14.19
Operating Cost, per Hour	USD/hr	$4.44
Total Cost of Ownership per Hour	**USD/hr**	**$18.63**

Source: SemiAnalysis AI TCO Model

RTX 6000 Pro offload scenario

The second offload scenario is to an RTX 6000 Pro Server Edition. For an apples-to-apples comparison, 96 robots require three 8-GPU servers, since the RTX 6000 Pro can serve four

Cost of Offloading Compute to GPU Server		
		RTX 6000 Pro
Server Cost	USD	
Server Service	USD	
Networking Cost	USD	
Software Licenses and Other Costs	USD	
Other Installation	USD	
Total Cluster Capital Cost	USD	
Wireless TX/RX Card	USD	
Total Capital Cost	**USD**	**$436,539**
Useful Life	Years	6
WACC	%	10.3%
Total Capital Costs per Month	USD	$8,153
Total Operating Costs per Month	USD	$5,460
Capital Cost, per Hour	USD/hr	$11.17
Operating Cost, per Hour	USD/hr	$7.48
Total Cost of Ownership per Hour	**USD/hr**	**$18.65**

Source: SemiAnalysis AI TCO Model

Jetson Thor on-device

The on-robot scenario is 96 Jetson Thor modules at $3,500 per unit, each with its own baseboard, storage, cooling, incremental battery, and onboard radio connectivity, landing at ~$394K. Accounting for the above, along with power costs for the Thor modules, leads us to a TCO of $14.97/hr for the 96-chip deployment.

We assume a four-year useful life for an on-device Jetson Thor, shorter than a datacenter GPU, because of the environment it lives in. A GPU in a rack never moves. A Jetson on a robot rides on a chassis that vibrates constantly, takes the occasional hard knock, and, depending on the job, gets dropped, splashed, or run through unstructured work all day. That mechanical stress wears hardware out faster. Some operators we spoke to put the figure below four years, but we think four is a fair assumption.

funded neoclouds or well-capitalized enterprises running their own servers. However, since we have already accounted for a shorter economic useful life for on-device compute as our base case, we have conservatively decided to keep a constant WACC across all three scenarios.

Cost of On-Device Compute

		Jetson Thor
Jetson Thor	USD	
Jetson Thor Baseboard	USD	
Local Storage (NVMe SSD 512GB)	USD	
Thermal Solution	USD	
Battery (Incremental capacity addition)	USD	
Onboard connectivity	USD	
Spares / RMA provisioning (~7%)	USD	
Total Capital Cost	**USD**	**$394,958**
Useful Life	Years	4
WACC	%	10.3%
Total Capital Costs per Month	USD	$10,074.2
Total Operating Costs per Month	USD	$856.1
Capital Cost, per Hour	USD/hr	$13.80
Operating Cost, per Hour	USD/hr	$1.17
Total Cost of Ownership per Hour	**USD/hr**	**$14.97**

Source: SemiAnalysis AI TCO Model

TCO Comparison

Before accounting for utilization, the three cases present differing TCOs per FP4 dense FLOPs: $0.15/hr/PFLOP for the B300, $0.16/hr/PFLOP for the Jetson Thor, and $0.39/hr/PFLOP for the RTX 6000 Pro.

The immediate conclusion that this might provide us is that the RTX 6000 Pro is not particularly performant on a FLOPs-per-TCO or a FLOPs-per-capex-dollar basis - we focus on the B300 offload scenario and the Jetson Thor on-device scenario from here.

Aggregate FP16 PFLOPs - Dense	PFLOPs	20	48	25
Aggregate Memory Bandwidth	TB/s	64	38	26
TCO per FP4 Dense FLOPs, before Utilization	USD/hr/PFLOPs	$0.16	$0.39	$0.15
TCO per FP8 Dense FLOPs, before Utilization	USD/hr/PFLOPs	$0.47	$0.39	$0.30
TCO per FP16 Dense FLOPs, before Utilization	USD/hr/PFLOPs	$0.93	$0.39	$0.60
TCO per Memory Bandwidth, before Utilization	USD/hr/TB/s	$0.29	$0.49	$0.57

Source: SemiAnalysis AI TCO Model

Utilization is the swing factor in TCO. A robot and the GPU behind it are almost entirely fixed capital, so the bill is the same whether the asset runs or sits idle, and cost per productive hour scales inversely with how much it works.

On-robot chip utilization today is highly deployment-dependent. In the home, one company that has deployed robots in volume told us its robots run just 1-2 hours a day, only 4-8% of the clock, because chores are bursty and current models still cap what can run unsupervised. That figure is trending up, toward 4-5 hours (17-21%), as capability improves and new features drive engagement. But the home has a hard ceiling: eventually the chores run out. Industrial demand is continuous and should clear a higher bar. Figure's BMW deployment is an anchor we have: roughly 1,250 operational hours over about 11 months of weekday shifts, working around 10 hours a day at 40% utilization. As model capability improves, industrial utilization should climb well beyond that.

Utilization under Various Deployment Scenarios

Deployment scenario	Hours/day	Utilization
Home deployment, low utilization	2	8.3%
Home deployment, medium utilization	5	20.8%
Home deployment, high utilization	10	41.7%
Industrial deployment, low utilization	10	41.7%
Industrial deployment, medium utilization	15	62.5%
Industrial deployment, high utilization	20	83.3%

Source: SemiAnalysis AI TCO Model

can serve 12 robots within the p99 latency budget, we model this as a ~90% utilization of the GPU server.

For on-robot chip utilization, we model ~40% utilization as our base case, which is factored into the table below. Net of utilization, the offload scenario for the B300 reaches a TCO per FP4 dense FLOPs of $0.17/hr/PFLOP, while the on-robot scenario for the Jetson Thor reaches a TCO per FP4 dense FLOPs of $0.37/hr/PFLOP. Here, the offload scenario reaches a mere ~46% of the on-robot scenario.

TCO per Performance Metrics

		B300 1200W	RTX 6000 Pro	Jetson Thor
Aggregate FP4 PFLOPs - Dense	PFLOPs	120	48	100
Aggregate FP8 PFLOPs - Dense	PFLOPs	40	48	50
Aggregate FP16 PFLOPs - Dense	PFLOPs	20	48	25
Aggregate Memory Bandwidth	TB/s	64	38	26
TCO per FP4 Dense FLOPs, before Utilization	USD/hr/PFLOPs	$0.16	$0.39	$0.15
TCO per FP8 Dense FLOPs, before Utilization	USD/hr/PFLOPs	$0.47	$0.39	$0.30
TCO per FP16 Dense FLOPs, before Utilization	USD/hr/PFLOPs	$0.93	$0.39	$0.60
TCO per Memory Bandwidth, before Utilization	USD/hr/TB/s	$0.29	$0.49	$0.57
Utilization Rate[1, 2]	%	90.0%	90.0%	40.0%
TCO per FP4 Dense FLOPs, after Utilization	USD/hr/PFLOPs	$0.17	$0.43	$0.37
TCO per FP8 Dense FLOPs, after Utilization	USD/hr/PFLOPs	$0.52	$0.43	$0.75
TCO per FP16 Dense FLOPs, after Utilization	USD/hr/PFLOPs	$1.03	$0.43	$1.50
TCO per Memory Bandwidth, after Utilization	USD/hr/TB/s	$0.32	$0.54	$1.43

1. GPU utilization rate is measured as number of robots served per GPU over maximum batch size, subject to a 90% cap.

2. On-robot utilization rate is highly dependent on robot deployment scenarios.

Source: SemiAnalysis AI TCO Model

Towards the top right corner of the sensitivity analysis below, high GPU server utilization and low chip on-device utilization swings the TCO per PFLOP massively in favor of the offloaded scenario. The offload scenario for current industrial deployments reaches ~46% the TCO per PFLOP of the on-robot scenario, reaching an even lower ~12% for home deployments.

It is only in several cases towards the bottom right corner wherein chronically low B300 cloud server utilization and chronically high on-device utilization swings the TCO per PFLOP in favor of on-device inference.

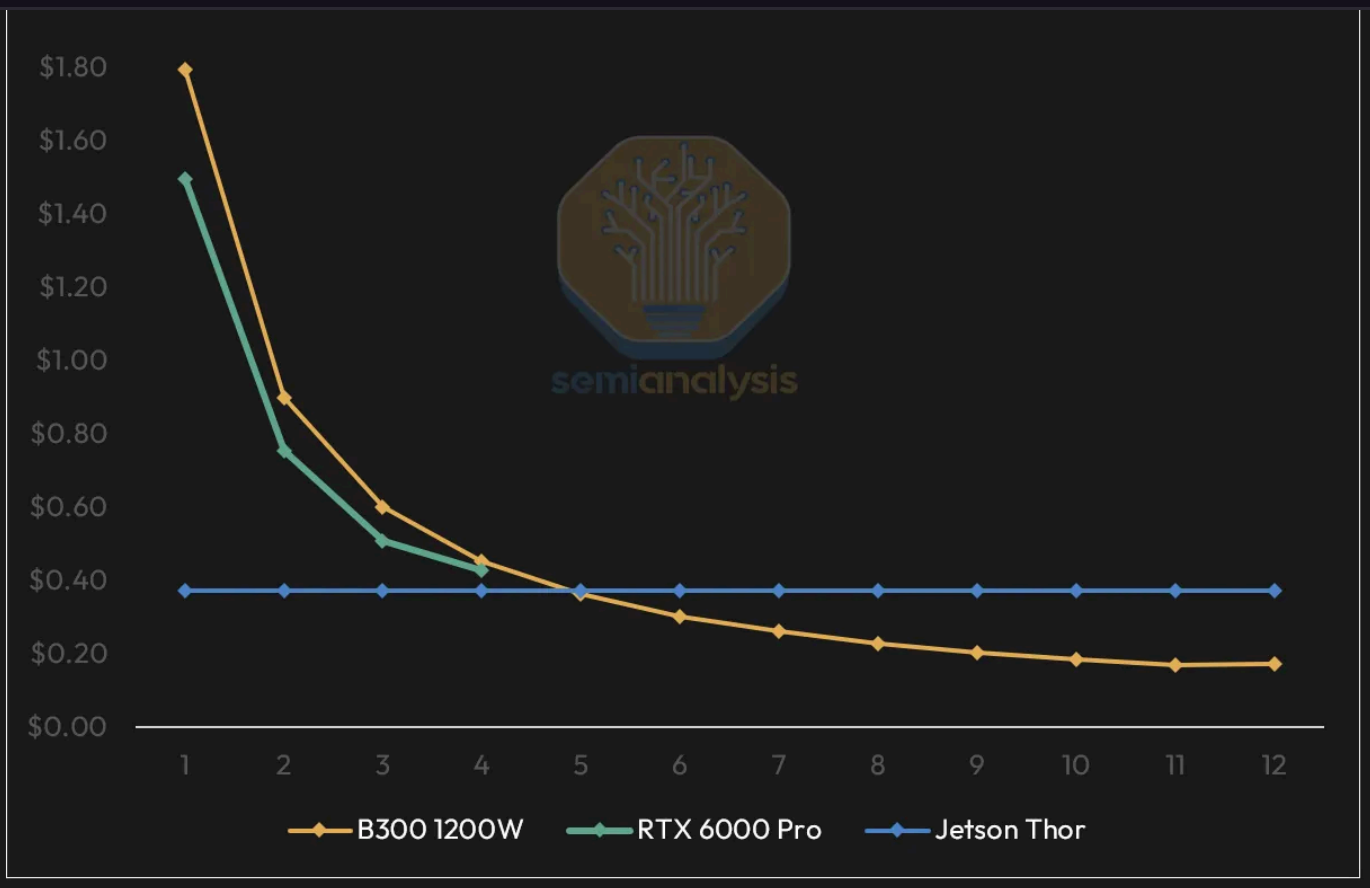
Utilization (Chip On-device)	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
10%	0.17x	0.15x	0.13x	0.12x	0.10x
20%	0.35x	0.30x	0.26x	0.23x	0.21x
30%	0.52x	0.44x	0.39x	0.35x	0.31x
40%	0.69x	0.59x	0.52x	0.46x	0.41x
50%	0.86x	0.74x	0.65x	0.58x	0.52x
60%	1.04x	0.89x	0.78x	0.69x	0.62x
70%	1.21x	1.04x	0.91x	0.81x	0.72x
80%	1.38x	1.18x	1.04x	0.92x	0.83x
90%	1.55x	1.33x	1.16x	1.04x	0.93x
100%	1.73x	1.48x	1.29x	1.15x	1.04x

Green indicates scenarios where offloading to a B300 is cheaper than on-device edge inference.
Red indicates scenarios where offloading to a B300 is more expensive than on-device edge inference.

Source: SemiAnalysis AI TCO Model

The question arises naturally: How many robots have to be batched in order for such TCO per PFLOP savings to come through in favor of the offload scenario? Varying our batching assumptions below, for a standard industrial deployment, we see that anywhere from 5 robots per B300 GPU and upwards, the TCO per PFLOP shifts in favor of the offloaded scenario.

On the whole, we find that offloading inference to a B300 server in the cloud is a compelling case, except for cases with very small robot deployments with one on-site server sitting mostly idle - though even then, one could make the argument for renting compute as needed rather than owning an entire server to serve a few robots.



1. Assumes chip on-device utilization of ~40%, as per current industrial deployments.

Source: SemiAnalysis AI TCO Model

The Deployment Battleground

On paper, off-robot compute wins on TCO. In practice, nobody is at mass production yet, so TCO is not what decides the question today. The companies actually shipping have split into two camps, those that keep compute on the robot and those that keep parts of it off, and what puts them in one or the other is how much generality the robot needs, how bad the wireless is, and how much of the customer's site they are allowed to change. TCO will matter once fleets get larger. For now, here is where each company stands today, and why.

Boston Dynamics

Boston Dynamics builds three robots: Spot walks inspection routes, Stretch unloads trailers and containers, and Atlas, the humanoid, entered production this year. Production here means development units built in volume for large-scale data collection and application pilots, not mass production. For Boston Dynamics, what matters most is capability,



Source: Boston Dynamics

Generalist vs Specialist

Most of those problems demand generality. A car plant is full of tasks that resist automation, from sequencing, kitting, and moving parts to line side, to assembly work like bolting a wheel onto the car. A single vehicle has tens of thousands of parts, one line can run 5 to 10 models at once in a dozen or more colors, and the mix turns over every model year. Commissioning a purpose-built solution for each step is neither fast nor economical at that scale. That variability is what caps traditional automation, and why factory work needs a generalist robot rather than thousands of specialist machines.



Source: Boston Dynamics. Hyundai Plant

Multimodality compounds the problem: because these machines operate in the physical world, System 2 has to consume images and reason over them, and models that do that well are inherently large. Boston Dynamics would not give an exact parameter count, pointing instead to Gemini Robotics ER, DeepMind's embodied reasoning model built on Gemini 3.5 Flash, as the right mental picture. Google publishes no parameter counts, but we estimate that System 2 could be somewhere in the hundreds of billions of parameters if not 1T.

A Brain Too Big to Carry

A model that size will not fit on a robot, which is why Boston Dynamics splits the stack across the network. System 1, the visuomotor policy that drives the machine, runs onboard, which on Atlas means an NVIDIA Jetson Thor. System 2, the reasoning layer, runs off the robot, reached through Orbit, the company's fleet software, and served on Google TPUs today through the DeepMind partnership. Of the companies we spoke to, Boston Dynamics is one of the few that deliberately puts high-level reasoning on the far side of a wireless link.

In a perfect world there would be no split at all. Ideally System 2 would sit on the robot with no wireless link in the loop, but no model small enough to carry can yet reason well enough for factory work, and the company would rather ship something intelligent now than wait. The gap may close in the future from stronger embedded silicon or from small models learning to reason well. For now, off-device is what works.

How Boston Dynamics defines System 1 and System 2 matters, so it is worth being clear. System 2 is the planner. It takes a work order from the manufacturing execution system, something like execute job 345 and put the result in inventory bin 5, and breaks it into steps simple enough to act on. System 1 is the VLA, the model that actually moves the robot: it takes a camera view plus one of those short instructions, pick up the can and put it into the bin with the green dot on it, and turns it into motor actions. It runs onboard because motion has hard latency and determinism requirements. Notice what happened between the two.

The factory's inventory bin 5 means nothing to a VLA, so System 2 marked the right bin with a green dot in the robot's view. That translation from the messy real world into terms System 1 understands is a core part of System 2's job, and the green dot is one technique from a fairly deep toolbox for steering what System 1 acts on. System 2 also supervises: it

the working assumption is a query every few seconds to every ten seconds, and in some cases once or twice a second

Another reason why System 2 cannot be on the robot is that the hardware it needs is datacenter hardware. A model that size wants a rack-mounted, liquid-cooled Blackwell-class GPU drawing 1.2 to 1.4 kW per chip. A humanoid carries roughly 2 kWh of battery and draws a few hundred watts in normal motion, a kilowatt or two at peak with everything included, which is why its onboard GPU is often a 40 to 130 W Jetson Thor. Bolt a datacenter GPU onto the robot and the chip alone would out-draw the entire machine, demand cooling the robot cannot carry, and flatten the battery in minutes.

The Tail-Latency Problem

Offloading System 2 allows you to use a frontier model, but the tradeoff is in networking, chiefly reliability and jitter. Average latency is less of an issue, the problem is the tail: the occasional one-second spike that freezes a robot mid-task. Once reasoning sits on the far side of a wireless link, networking becomes the single biggest problem in robotics, and Boston Dynamics has multiple teams dedicated to nothing else, covering the radio stack on the robot, the customer's infrastructure, spectrum, and protocol design. We go into that networking portion in detail in the next section.

Whose Data Is It

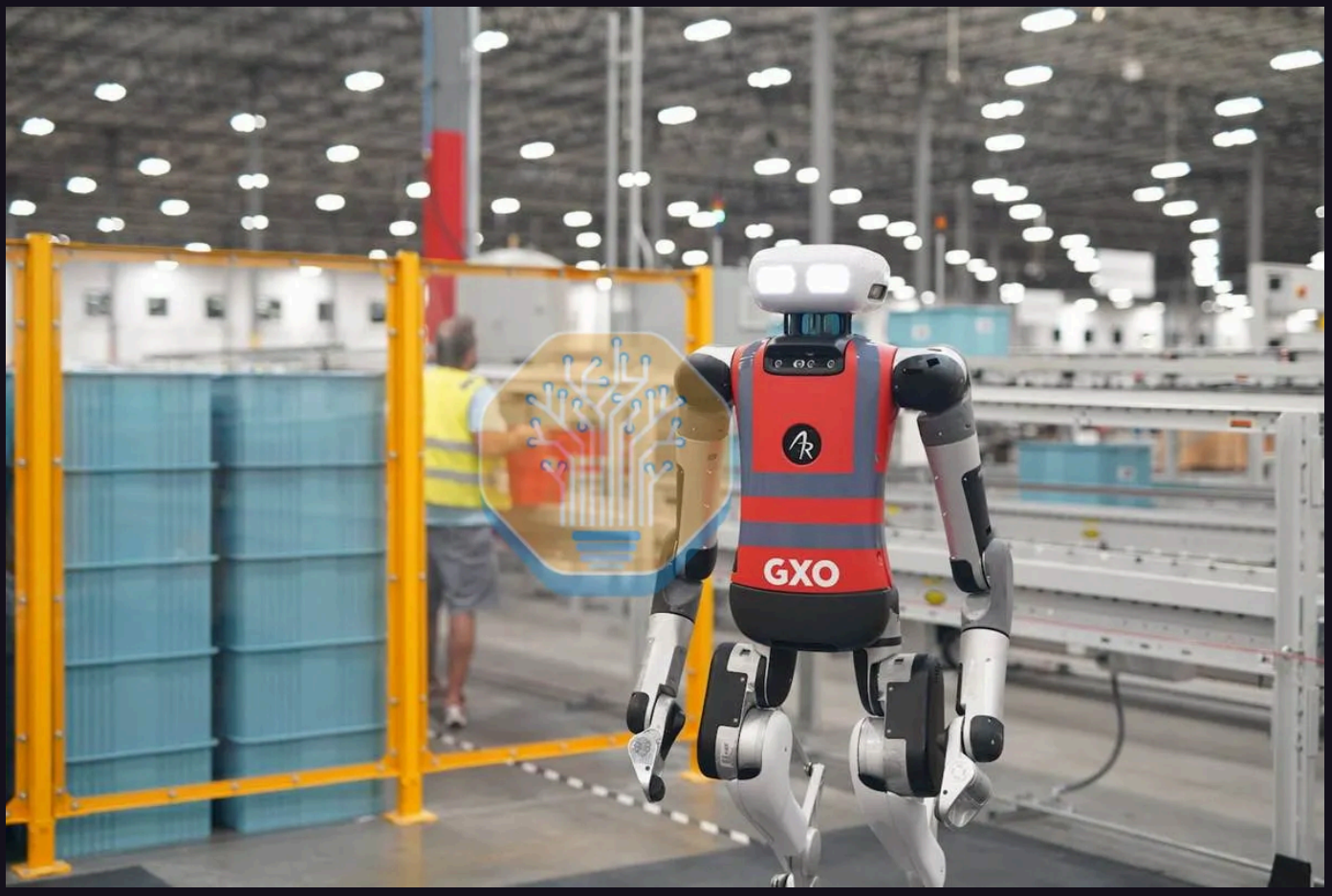
The other consideration of offloading System 2 over a wireless link is data security. Every robot is effectively exporting a video feed of someone's factory floor, and customers rightfully attach conditions: no training on their data, no mixing it with other customers', deletion on demand. Compliance is fragmented as well, with each jurisdiction bringing its own quirks in data protection law. And some customers, like nuclear plants or military operators, will never let data offsite under any guarantee.

Boston Dynamics' answer is layered, following the data up the pipeline. It starts on the robot, where customers get granular control over what is shared at all. What does leave lands in Orbit, the cloud platform the robots report to, which carries SOC 2 Type 2 certification and sits under an internal data committee that governs handling. The last hop is Google: System 2 runs on Google infrastructure through the DeepMind partnership, so every image a robot sends up for reasoning passes through their side as well. That half is

The company's bet is that customers will accept offsite data once you can solidly prove it's treated correctly, the same logic that lets sensitive enterprises send queries to Anthropic under no-training guarantees.

Agility Robotics

Agility Robotics builds for the warehouse and the factory floor. Digit, its bipedal humanoid, moves totes, tends machines, and palletizes. Its customers include GXO, Schaeffler, Amazon, Toyota Motor Manufacturing Canada, and Mercado Libre. Agility does not train base models from scratch. Instead, it takes a performant third-party base, and post-trains on top of it. Since the language and action conditioning comes from Agility's own corpus, there is no reason to train a vision model from zero. All of this fits onboard the robot on an NVIDIA Jetson-class accelerator.



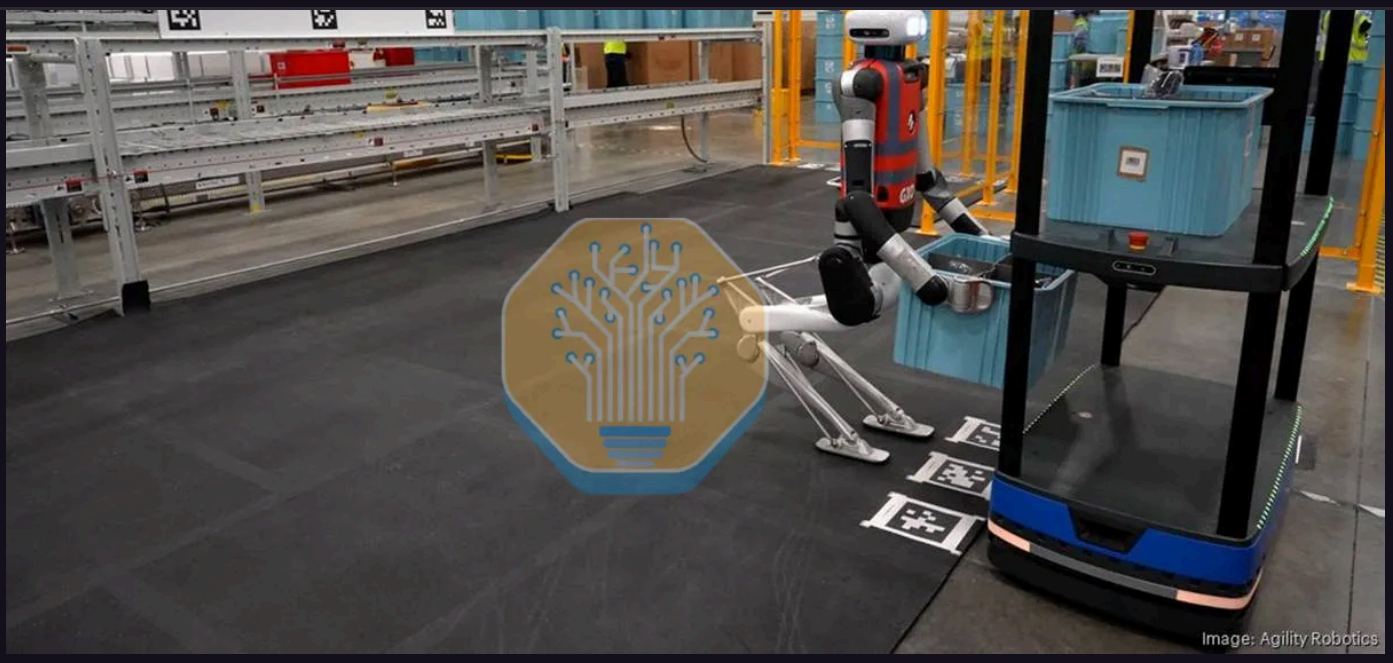
Source: Agility Robotics

Even with inference happening onboard, Agility still offloads some functions to the cloud. Fleet orchestration, workflow assignment, site mapping, and over-the-air updates all live in

can go ten seconds without receiving its next workflow chunk because it already has thirty to sixty seconds of queued work in front of it, so it makes sense for that layer to sit in the cloud.

For everything else, Agility is firmly in the GPU-on-robot camp. Network reliability and latency are real concerns, and factories are especially tough environments for wireless connections: moving metal, heavy machinery, multipath interference, and a robot that drops line of sight every time it passes a shelf, roaming between access points as it crosses the floor. It is possible to engineer around all of this and get a network reliable enough to offload compute. But that means reworking a customer's network and edge infrastructure, and Agility wants Digit to slot into existing sites with as few changes as possible.

Networking is not the only thing keeping compute on the robot. Safety pushes in the same direction, and it is an area where Agility has built an industrial-grade stack that many humanoid competitors lack. Today, Digit operates exclusively inside physically barricaded work cells, as no dynamically stable robot is yet safety-certified to work in close proximity to people. A balancing biped requires active control to stay upright and could topple if power is lost, so OSHA-regulated environments mandate physical separation between Digit and human workers. Digit's sensor suite provides 360-degree awareness through cameras and LiDAR, letting it detect people and halt or adjust its path in real time. The next generation goes a step further, adding a dedicated human detection system meant to recognize an approaching person, stop, and lower itself to the ground before contact is possible, clearing the bar for Digit to eventually work outside the barriers. Digit also carries other industrial safety features, such as a stop function that maintains power to the actuators during deceleration, so the machine slows smoothly and safely before power is cut.

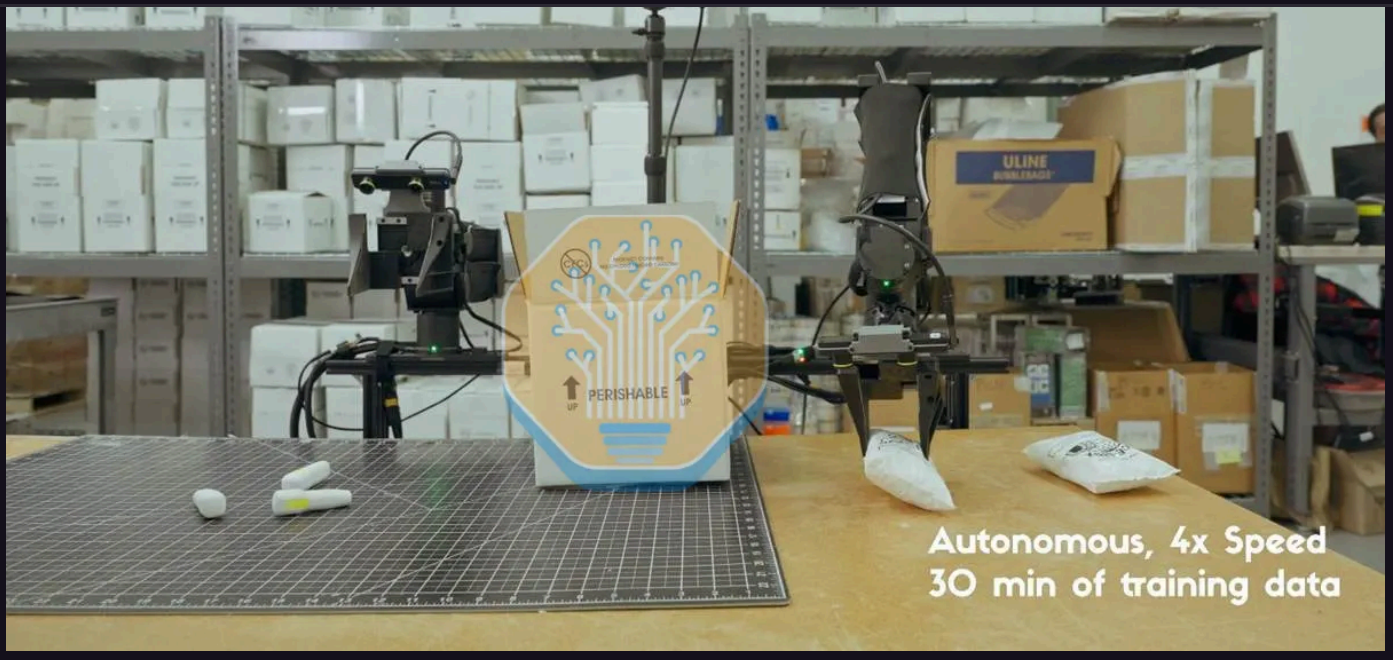


Source: Agility Robotics

All of these precautions exist with compute fully onboard. Moving safety decisions off the robot would create a more fundamental problem: the network link itself becomes a failure mode. If the robot loses connection to an offboard safety model, it cannot tell whether anything is wrong, so its only option is a blind, worst-case fail-safe. That is why complex safety decisions must always stay on the robot.

Verne Robotics

Verne Robotics is deploying its robots primarily in warehouses and light manufacturing environments. Its robot, Nemo, is a dual-arm mobile manipulator consisting of a bimanual upper body mounted on a wheeled base, available with either 700 mm or 1,000 mm reach arms. Nemo has been deployed at logistics companies in Massachusetts and California, including AbClonal, a life-science reagents company, where it automates cold-chain workflows such as retrieving, packaging, and transporting chemical reagents and ice packs. Verne is also expanding deployments into light manufacturing, where utilization rates can be even higher.



Source: Verne Robotics

Notably, rather than relying solely on teleoperation, Verne uses proprietary camera-equipped data collection suits worn by employees and contractors to capture embodied data directly at customer sites. The company develops its own end-to-end policy models, likely in the few-billion-parameter range, that run entirely on Jetson-class or RTX 50-series GPUs. Because warehouse manipulation and light manufacturing tasks such as picking, packing, and assembly do not require broad open-world reasoning, Verne achieves high task success without relying on large cloud-hosted foundation models. The entire perception-to-action stack runs locally on the robot, eliminating the need for offboard inference or datacenter connectivity during execution.



Source: Verne Robotics

Sunday Robotics

Sunday Robotics builds for the home. Memo is a wheeled machine whose torso rises and lowers to reach from the floor to high cabinets, with three-fingered grippers instead of humanoid hands. Sunday also trains their own foundation models. Instead of teleoperation, Sunday collects training data through its Skill Capture Glove, a wearable that matches the geometry of Memo's gripper, worn by people simply doing their chores at home. That glove data, combined with other commodity data such as videos, pretrains one end-to-end policy, and post-training polishes it.



Source: Sunday Robotics

Over the past 12 months, Sunday has released two models, each bringing Sunday closer to a system that can perform any task in any home. Released in November 2025, ACT-1 focused on breadth of capability. It enabled Memo to complete a 15-minute, long-horizon task of clearing a dining table and loading everything into the dishwasher, while also demonstrating high-dexterity skills such as folding socks and making espresso, as well as generalizing to previously unseen homes.

ACT-2, previewed in July, turns that breadth into reliability: 99.1% zero-shot success folding laundry across nine garment types in homes the robot had never seen. The finding Sunday cares most about is that scaling pretraining is what makes post-training generalize, so much so that one demonstration can teach a new behavior that transfers. The chart shows how stark this is. In-domain means Sunday's own lab, where the post-training data came from; out-of-domain means held-out homes and objects. Without pretraining, the model hits 96% in the lab and just 14% in unseen homes, textbook overfitting. With the full pretraining corpus, the gap disappears: 100% in both.

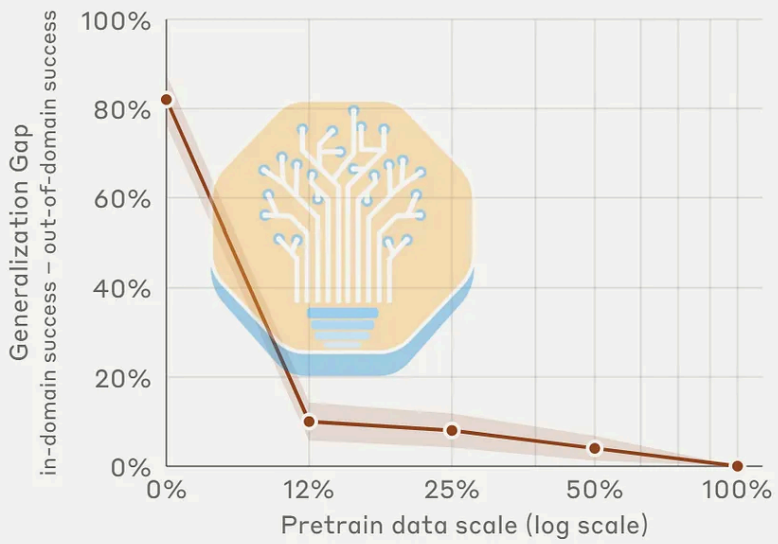


Figure 1 · Generalization gap between in-domain and out-of-domain folding success across pretrain data scale; the shaded band shows ±1 standard error, with 50 evaluations behind each success rate.

Source: Sunday Robotics

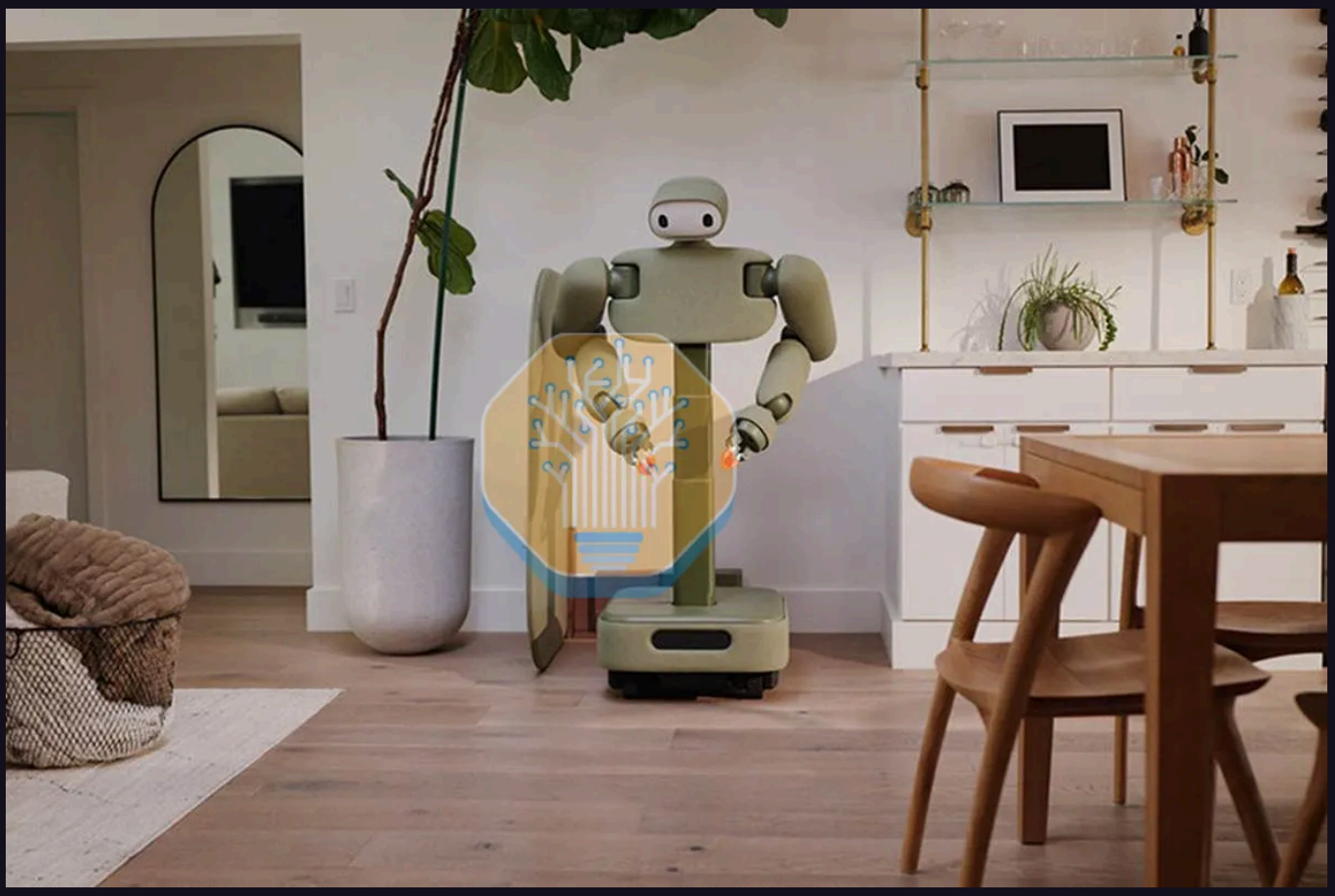
Generality comes from pretraining scale, and pretraining scale needs a model large enough to absorb it. The policy that works in a stranger's laundry room is not a small one. So Sunday started out assuming cloud inference, and in the lab that made sense: more compute, easier debugging, and the same setup the researchers already worked with. Two days into testing ACT-2 in real homes, they reversed course. Home WiFi turned out to be a mess of dead zones, neighbor interference, badly configured mesh routers, and asymmetric links. Latency was mostly fine. Jitter was the problem. A robot trying to hold 99% success over hours can't shrug off the occasional hiccup the way a video call can. The alternatives didn't help. Starlink's satellites cross the sky in minutes, so any tree or roofline inside the dish's 100-degree view knocks the link out on every pass, and a 5G router dropped out the same way.



Source: X, Sunday Robotics

Sunday faced a fundamental tradeoff: larger models generalize better, but relying on the cloud makes the robot dependent on network connectivity. Sunday chose to resolve that tradeoff by moving inference fully onboard. The model now runs directly on the GPU inside the robot, removing the cloud from the critical path and allowing Memo to operate independently of the network.

folds laundry you bring to it. Isaac 1, due this fall, keeps all of that and adds a wheeled base and a telescoping torso, so it can go find the dirty clothes itself, put the folded ones away, make beds, and get toys and shoes back where they belong. Weave trains its own VLA and also runs models from Physical Intelligence. They build their own model to stay as close to the frontier as possible, but will use whatever gives the customer the better experience, so this is not a Pi wrapper and it is not a closed shop either. Both are likely in the few billion parameter range, small enough to sit on the robot, and we would expect inference to run locally.



Source: Weave Robotics

Even if inference runs on the robot, this does not make Weave free of the network. The WiFi link carries proprioceptive and visual data goes up as training data, and when the robot encounters difficulties a teleoperator steps in and takes control. Both degrade with the link, and a bad enough connection kills traffic in both directions. In homes, walls and other obstacles create weak spots and dead zones, and the typical router is not built for handoffs. Conditions are often good, but congestion is a factor: homes usually run on

To deal with these conditions, Weave does not require any changes to the home environment: no new router, or no extra access points are required. That pushes all of the adaptation onto the robot.

Climbing the Networking Wall

Many of today's deployments keep inference on the robot, and nearly every company gives the same reason: the network. For the companies that deploy off-device, they got there by staffing multiple teams that work on nothing else. We think the networking problems are solvable, and that as they get solved, more deployments will move off the robot.

What is the Network Wall?

Running a robot model off-device is extremely difficult (see image below). Three high-level issues make off-device policies challenging.

Embodiment. At the physical layer, you are asking for good network conditions from a machine that is made of metal, that moves, and whose motors and electronics emit noise on the same frequencies you're trying to transmit on. The chassis blocks and reflects its own signal, orientation changes antenna positions, and RF conditions are dynamic between transmissions.

Network. The infrastructure around WiFi and 5G was not designed with robotics in mind. It works well at home, with one or a few devices, streaming Netflix while you sit on the couch. But robotics is the opposite. You still have a Netflix-sized video feed, but you're streaming it to the datacenter, and you're moving while you do it. This is the uplink problem: all of the existing infrastructure was built with the downlink in mind.

The occasional dropped packet or latency spike is fine for Netflix, but an accident waiting to happen in robotics. Existing access points, chipsets, and algorithms simply weren't built for it: prioritization is inverted, uplink MIMO is rarely supported, no redundancy to make UDP streams reliable, no low-latency handoff between access points, uplink scheduling is an afterthought, and everything is statistical when it needs to be deterministic. Add shared fiber to the picture, and your neighbor turning on Netflix while your humanoid is working can, quite literally, kill you.

area is enormous, the operator has almost no control over RF conditions, and the vehicle moves fast, which leaves near-zero latency budget and forces frequent access point handoffs. Add that driving is a simpler learning problem than humanoid labor, so the models fit on-device, and the conclusion follows: compute goes on-device, and the off-device stack shrinks to fleet management, a slow, low-frequency loop.

The environmental prerequisite for off-device compute is a small-to-medium sized space that is structured enough for RF control. Autonomous driving lacks this; homes and factories have it. That doesn't make it easy – the environmental problem is still huge. Walls, metal, and other electronics all affect RF conditions. Turn on the dishwasher at home and the RF environment changes. Start heavy machinery in a factory and you may create a dead zone beside it. Access point placement matters a lot, and it has to anticipate how the environment will change over time. Moving between access points means handoffs, and a typical router goes silent for anywhere from 100 ms to several seconds during one. That can happen mid-manipulation, killing a policy or lagging a teleoperator. Recovery can take several more seconds, and by then the humanoid may be on the floor.

Placement of signal blockers and reflectors in the home is unpredictable; every house is a different RF puzzle. A factory has more structure but far more working against you: dense metal racking that reflects and blocks signal, motors and machines spraying interference across the spectrum, inventory and equipment that move daily so the dead and weak zones move with them, and dozens of robots contending for the same uplink airtime.

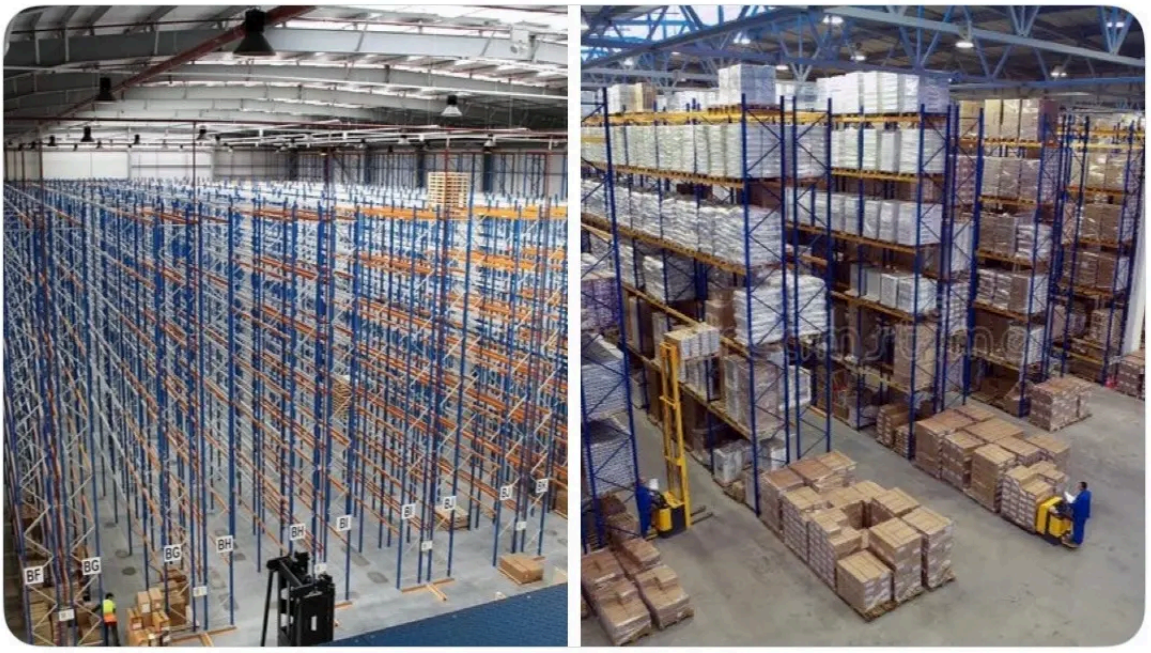
my man has obviously never tried to connect to wifi/5G on a factory floor

> **Gianluca** @gianlucabencomo · 5d
> Without a doubt:
>
> A world running foundation models on a billion robots won't be doing it locally....

6 2 70 12K

Jack @JacklouisP · 4d

Searching for bars in this



6 176

Source: X

How do we climb it?

perform, its action space, and the embodiment it drives. Some of what breaks today is fixable with ordinary engineering. The transport layer, for one, is often a modified WebRTC: UDP with no retransmission, reliability through repetition over multiple antennas, application-layer awareness of packet importance, and freshness above all. Teams already build this and the know-how is a commodity.

But the network wall is not the transport layer or a series of ordinary engineering challenges. The network wall is based on the issues with existing hardware, and we believe there are four problems that are hard and high-leverage that together can make off-device models reliable beyond the existing infrastructure.

Start by fixing the environment, not the robot. The highest-leverage fix is the access point. Access points in robotics need to do the opposite of what consumer and enterprise access points do for downlink-heavy stationary clients. This includes:

- **Robot-aware uplink scheduling.** Robot uplink is scheduled ahead of household or plant traffic rather than queued behind it, which is admission control that WiFi does not natively have and the access point must impose. Each robot gets a reserved slot at a fixed cadence instead of a chance at the channel, and the schedule accounts for the other robots competing for the same airtime. The model determines the traffic, the traffic determines the schedule, so the scheduler has to know what the robots are running. When a robot is not phoning home, the same scheduler still has to manage everything else on the network well.
- **Location-aware beamforming.** Robots move. The access point has to know where each robot is and where it is going, then steer, schedule, and hand off ahead of that movement rather than reacting to it or assuming a client that stays where it is associated.
- **Centralized timing.** One timing authority per site, distributed over the air to every robot with a guarantee, so observations across the fleet share a clock. Off-the-shelf access points cannot provide this.
- **Multi-link operation.** Running several links at once, across bands and across WiFi and 5G, with the same observation duplicated or split across them, improves reliability and cuts latency in a way no single link can.
- **Clean spectrum.** 6 GHz where available, since it is uncongested and its channel widths give the throughput margin that makes retransmission-free delivery realistic.

handoff is multiple dropped frames.

Placement is just as important as the hardware. It needs to be measured and optimized, not guessed: RF survey along the robot's task paths, dead-zone mapping, antenna orientation chosen for the robot's typical height and heading.

WiFi will dominate the home and most of the factory floor because it is fast and can carry the load. It will not cover everything. 5G, private or public, fills outdoor paths, larger sites, and locations where WiFi congestion cannot be controlled. The two should coexist under one scheduler, with the robot able to fail over fast between them. Where a site depends on a carrier, the last step is to work with the network provider on uplink prioritization and dedicated fiber to the GPU cluster, so that the scheduled path does not end at the access point.

Shrink the load. Networks are bandwidth-limited. The other half of climbing the wall is reducing what has to cross it. Image sensors are nearly all of a robot's uplink, so this is a perception problem before it is a networking problem.

Maximize the uplink. Robotics inverts the traffic pattern every wireless chipset was designed for. The mainboard has to be built for this. 4x4 uplink MIMO rather than the one or two streams consumer modules ship, chipsets chosen for uplink throughput rather than downlink, an operating system built for wireless communication, and a radio that participates in the access point's schedule instead of contending with it: it knows its slot, fills it at cadence, and stays quiet outside it. Holding that beat is what makes scheduling deterministic, and it only works if the mainboard and the access point are co-designed. Neither side can impose order on the link alone.

Everything on one clock. Datacenter GPUs beat on-device compute on cost as batch efficiency grows, and batching requires that observations from many robots arrive time-aligned in order to hit a reasonable latency budget. Everything has to run on the same clock. Sub-millisecond synchronization is achievable with good crystal oscillators on the mainboard and wireless time sync across the fleet, so every robot on a site captures, encodes, and transmits on the same beat. GPU orchestration is then built around that beat: the scheduler knows when the next frame of observations lands, reserves compute for it, and runs the moment the frame is complete. Late observations go to the next batch, not to a queue, and the latency budget is spent on inference rather than waiting. Put this together with a mainboard that holds the schedule, access points that enforce it, and a perception

Who will climb the wall?

This post is for paid subscribers

Subscribe

Already a paid subscriber? Sign in

← Previous

© 2026 Dylan Patel · Privacy · Terms · Collection notice

Start your Substack

Get the app

Substack is the home for great culture