Architectural Fit for Production-Scale Agentic AI on Heterogeneous SoCs

Siavash Alamouti, Fay Arjomandi, Michel Burger, Jeremy Hsu

mimik Technology Inc.

Abstract

Agentic AI on end-device silicon differs structurally from the model-inference benchmarks that have shaped on-device AI hardware. Multi-agent stacks spend most of their work not on inference but on operational work, the data management, I/O, inter-agent collaboration, security, discovery, and routing that runs on the CPU and scales with agent count rather than model size. We model agentic workloads on heterogeneous SoCs across agent count, workload mix, operational overhead, model size, and platform, comparing two architectural classes, a CPU-centric SoC and a GPU-centric SoC, with the AMD Ryzen AI Embedded X100 as the CPU-centric example and an NVIDIA Jetson Thor-class part as the GPU-centric example. The CPU-centric class holds the most CPU headroom in 100 percent of modeled scenarios. On total compute headroom it leads in 74 percent of scenarios under static single-engine placement and in 82 percent once an agentic operating environment balances each workload across the available engines, winning every sensing-heavy and balanced workload and ceding only the large-model reasoning-heavy workloads to the GPU-centric class, whose larger GPU carries that work with more headroom. Per device the CPU-centric class sustains roughly 2.25 times the concurrent agents of the GPU-centric class, a CPU-bound advantage, at comparable energy. The conclusions hold across Device-First Continuum AI deployments, with physical AI as the principal focus.

1 Introduction

The shift from classical control and inference stacks to agentic AI architectures has reshaped the computational profile of end-device platforms across a wide range of application domains. In classical stacks, perception, reasoning, and action execute as a tightly coupled pipeline against a fixed task definition. Current agentic stacks largely run as monolithic processes that bundle multiple agents into a single application binary, an approach that becomes increasingly difficult to scale, update, and operate as agent counts grow. The Agentix-native[1] approach proposed in prior work [5], and analyzed in this paper, organizes each functional capability as an autonomous microservice that reasons about its own state, escalates when it reaches its limits, and coordinates with peer agents through service discovery and message passing. The result is a stack that handles goal-level instructions and adapts to environments it was not explicitly programmed for, at the cost of substantial new operational work.

This paper analyzes the workload distribution of agentic stacks on heterogeneous SoCs. We are concerned specifically with the agentic tier, which sits above the operating system and alongside the application's real-time or domain-specific control stack. The agentic tier provides goals, context, and reasoning through a defined interface to the domain layer below it. The agentic tier is where the operationally driven CPU workload growth that has motivated recent end-device silicon designs principally lives, and it is the tier on which this paper focuses.

[1]The "x" in Agentix denotes a cross-platform design: the same microservice agents run across CPU-, GPU-, and NPU-centric devices, from compute-capable end devices such as smart cameras and robots up to servers.

The analysis framework presented here applies across Device-First Continuum AI deployments that share the common structure of multi-agent coordination layered over TinyML, sLLM, and LLM inference, including physical AI and robotics, ambient healthcare and preventive monitoring, intelligent spaces and security, and fleet and asset management. The structural patterns that drive workload distribution (agent count, operational overhead, model size mix) are common across these domains. The numerical results in this paper take physical AI as the principal application focus, reflecting its current prominence in heterogeneous SoC roadmaps and in the silicon platforms under comparison.

We make three contributions. First, we develop an analytical model of the agentic tier as a function of five parameters that characterize the workload and the silicon platform. Second, we define two architectural classes of heterogeneous SoC, CPU-centric and GPU-centric, by their compute distribution and apply the model to a representative platform from each, using public silicon specifications and published per-operator profiling. Third, we present measured workload examples from existing agentic deployments as anchor points within the modeled space, demonstrating that real workloads fall where the analytical model predicts.

It is worth surfacing the deeper structural reason this matters. The shift from monolithic AI to agentic AI is, at root, a shift in compute distribution away from GPU dominance toward CPU-led work with greater NPU exercise. This shift does not arise from latency or operational overhead in any conventional sense; it arises from the nature of distributed agentic systems themselves. Almost everything a multi-agent system does besides model inference is CPU work: inter-agent communication (serialization, network stack traversal, deserialization), zero-trust security (authentication, encryption, message signing), agent and resource discovery, observability and distributed tracing, distributed state management, routing and orchestration decisions, pre- and post-processing (tokenization, output formatting, schema validation), and policy enforcement (PII detection, capability checks, output safety filtering). The GPU is consulted only when a model is actually being inferenced, which constitutes a smaller fraction of total work in agentic systems than in monolithic ones. This first driver is reinforced by a second: when reasoning is distributed across many specialized agents, each agent typically runs a smaller model than a monolithic stack would, and at small model sizes CPU and NPU become competitive with GPU per inference [2, 3]. A third driver compounds both on the device side. In physical AI deployments, in ambient sensing, in fleet and intelligent-space deployments, the device form factor constrains the size of any model that can reside on it, and the spatial and functional structure of the system mandates that agents run on many separate nodes coordinating over a local network. The compute distribution of on-device agentic AI therefore differs structurally from the monolithic LLM serving workload that current end-device silicon was designed around, and the difference is dictated by the architecture of the system rather than chosen as an optimization.

This progression is best understood as a sequence of two structural transitions, each shifting compute distribution away from GPU dominance toward CPU and NPU. The first transition, from monolithic AI to agentic AI, introduces a structural CPU floor from inter-agent communication, security, and discovery, with a move toward smaller per-agent models. The second transition, from agentic AI to on-device agentic AI, amplifies the first: device form factor constrains model sizes further, and the spatial structure of the system mandates that agents run on multiple nodes coordinating over a network.

Compute Resources on Heterogeneous SoCs

A modern heterogeneous SoC integrates three kinds of AI compute on a single die. We summarize each briefly because the agentic workload analysis depends on how work is distributed across them.

The **CPU** (central processing unit) is the general-purpose engine. It executes program logic, coordinates between components, handles control flow, and runs small or irregular com-

putations. In agentic AI, the CPU runs agent runtime code, routes messages between agents, handles authentication and lifecycle, and executes small language model inference where the workload is too small to keep an accelerator busy.

The **GPU** (graphics processing unit) is a massively parallel engine optimized for large regular computations on dense data. It excels at the matrix multiplications that dominate large neural network inference, particularly when the model is big enough that data transfer overhead is small relative to compute. In agentic AI, the GPU handles vision processing for complex perception, large language model inference when models exceed several billion parameters, and other compute-dense kernels.

The **NPU** (neural processing unit) is a fixed-function accelerator optimized for low-precision integer matrix operations that dominate quantized neural network inference. It is more power-efficient than the GPU for the workloads it handles well but less flexible: many models do not map cleanly to the NPU's supported operations. In agentic AI, the NPU handles continuous TinyML inference (sensor classification, anomaly detection, keyword spotting) and small model decode where INT8 quantization is acceptable.

The relative throughput each platform delivers across these three resources defines its architectural character.

Two Architectural Classes

We describe heterogeneous SoCs by the share of their total AI compute throughput that comes from each resource. For comparison, CPU, GPU, and NPU contributions are normalized to a common throughput scale (see Section 5). Two classes capture the dominant architectural philosophies relevant to this comparison in the embedded and physical AI market today:

- **CPU-centric.** A strong general-purpose CPU complex carries the operationally intensive portion of the workload, alongside a capable GPU and an NPU that each carry a meaningful share of inference throughput. No single resource is a weak point. Targeted at workstation-class embedded deployments where the silicon must handle a wide range of agentic workloads concurrently. The AMD Ryzen AI Embedded X100 is a representative example and is named explicitly in this paper.
- **GPU-centric.** A dominant GPU with Tensor cores carries most of the throughput; the CPU is a supporting resource and there is no separate NPU, since the GPU carries quantized inference directly. Targeted at accelerator-bound workloads typical of single large-model vision and language serving on end devices. An NVIDIA Jetson Thor-class part is a representative example and is named explicitly in this paper.

Figure 1 shows the compute distribution of one representative platform from each class. The X100's distribution is spread across CPU (45 percent), GPU (39 percent), and NPU (16 percent of normalized total), with no resource left as a weak point. The GPU-centric class concentrates throughput in the GPU (94 percent), with a small CPU share (6 percent) and no separate NPU, since its GPU carries inference directly.

The thesis of this paper is that agentic AI workloads exercise CPU, GPU, and NPU together, and that platforms whose compute distribution matches this profile handle the workload without bottlenecks. Platforms with distributions skewed toward one resource encounter saturation on whichever resource they underprovisioned for the workload.

Architectural Scope: Microservice-Based Multi-Agent Deployment

The analysis in this paper assumes a microservice-based multi-agent architecture. Most current agentic implementations in production are monolithic in deployment: orchestration frameworks such as LangChain, AutoGen, and CrewAI run all agents in a single process with shared memory

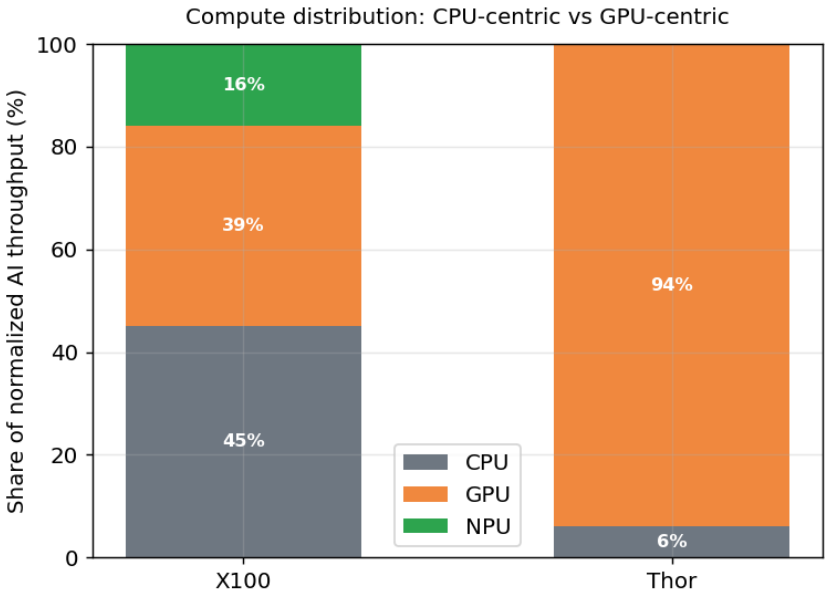

Figure 1: Compute distribution by architectural class. CPU, GPU, and NPU shares of total normalized AI throughput for one representative platform from each class.

and a centralized orchestrator. The monolithic approach offers marginal raw efficiency advantages on a per-inference basis but does not provide the primitives that multi-agent on-device systems need at fleet scale.

Prior work establishes the case for microservice-based agents quantitatively [5]. The Agentix-native paradigm packages each agent as an independently deployable microservice (a *mim*) running on a lightweight operating environment that provides dynamic service and resource discovery, lifecycle management, zero-trust security, and peer-to-peer communication. In a representative scenario analyzed by the cited work, the microservice approach showed substantial advantages over the monolithic app-native baseline along several dimensions: reductions in update payload size, storage savings from model deduplication, lower regulatory verification effort for safety-critical applications, smaller failure blast radius across the agent population, and lower expected memory than monolithic deployment at moderate agent counts due to serverless lifecycle behavior. The exact magnitudes of these improvements depend on the workload mix, agent count, and operational parameters; we refer the reader to the cited work for the specific measurements. The general point is that the microservice substrate provides architectural primitives that monolithic deployment cannot, and these primitives become more valuable as agent counts and fleet sizes grow.

The same prior work argues that the Agentix Operating Engine (mimOE), mimik's microservice-based substrate for multi-agent systems, is not an optimization but a prerequisite for any multi-agent system that must communicate beyond its own application process, which is the case for nearly all production deployments. We adopt this architectural substrate as the foundation of the present analysis. The silicon comparison that follows takes the microservice-based deployment as given, and the operational overhead parameter ρ in Section 3 reflects the inter-microservice communication characteristic of this architecture. For monolithic deployments, the structural conclusions hold qualitatively but the operational overhead is structured differently and the deployment lifecycle advantages discussed in Section 10 do not apply.

The remainder of this paper is organized as follows. Section 2 reviews related work on agentic LLM serving and on-device inference characterization. Section 3 decomposes the agentic workload into three tiers and characterizes the operational structure. Section 4 presents the analytical model. Section 5 parameterizes the three reference platforms. Section 6 presents

simulation results across the five-dimensional parameter space. Section 7 situates measured workload examples within the modeled space. Section 8 evaluates four use cases drawn from published deployments and a healthcare product against the three architectural classes and derives general observations about architectural fit. Section 9 extends the framework to cluster-scale deployment, where statistical pooling of capacity across multiple devices compounds the single-device architectural fit advantages. Section 10 discusses the deployment lifecycle dimension that microservice-based agent runtimes uniquely address. Section 11 summarizes the implications for platform selection across workload classes.

2 Related Work

Recent research on agentic LLM workloads has identified the structural mismatch between consumer-grade heterogeneous SoCs and the execution patterns of personal agents. Wei et al. [1] characterized agentic LLM workloads on Intel Core Ultra SoCs and observed that LLM engines designed for isolated single-shot inference are not well-suited to managing the concurrent reactive and proactive flows that characterize on-device agents. Their profiling exposed operator-accelerator affinity asymmetries, asymmetric DDR contention between CPU, integrated GPU, and NPU, and stage-divergent batching behaviors that differ from cloud-serving assumptions.

The assumption that on-device LLM inference is necessarily GPU-bound has been challenged empirically. Zhang and Huang [2] demonstrated that on the iPhone 15 Pro, a 1B parameter LLM achieves 17 tokens per second on the CPU (two threads, FP16) versus 12.8 tokens per second with GPU acceleration. The CPU advantage arises because GPU memory transfer overhead dominates at small model sizes. This finding extends to the embedded space where similar small-model regimes prevail for on-device reasoning agents.

Broader characterization of mobile LLM inference [3] documented that ARMv8-A CPUs (Cortex-A76 and A77) achieve 2 to 4 tokens per second on decode for typical 7B models, while ARMv9-A CPUs with specialized instructions deliver substantial gains. The Cortex-A78AE used in the GPU-centric class sits in the ARMv8.2-A class and inherits the corresponding throughput characteristics.

On NPU-accelerated inference, the analysis of mobile NPU LLM serving [4] established that prefill dominates 94 to 98 percent of end-to-end latency in agentic UI automation tasks, with each prefill step taking 8.1 seconds for Qwen1.5-1.8B on commodity mobile NPUs. This finding shapes how the analytical model in Section 4 weights the per-tier compute contributions.

On the architectural side, Alamouti, Arjomandi, Burger, and Hsu [5] analyzed two paradigms for organizing on-device AI microservices, comparing monolithic app-native deployment with serverless microservice-based (Agentix-Native) deployment on identical silicon. Their simulation framework, on which the present model builds, parameterizes runtime overhead, memory behavior, and inter-agent communication cost.

3 Workload Decomposition

We model an agentic stack as a collection of N concurrent agents, where each agent is classified into one of three tiers based on the size and structure of the inference workload it performs:

- **TinyML tier (T):** Continuous, low-latency sensor or signal processing. Models in the 1 to 10 MB range, quantized to INT8 or smaller. Examples include anomaly detection on inertial or biometric signals, gesture or keyword recognition, low-level object classification, and event triggers. TinyML agents run continuously at frequencies of 1 to 10 Hz per agent and are NPU-eligible.
- **sLLM tier (S):** Small language model reasoning for local task decomposition, coordination, and routing. Models in the 0.5 to 3 billion parameter range. Prefill phase is compute-bound

and CPU-favorable at small model sizes. Decode phase is bandwidth-limited and CPU- or GPU-eligible depending on memory architecture. Invocation pattern is event-driven, with rates ranging from 0.1 to 1 Hz per agent.

- **LLM tier (L):** Larger model reasoning for heavy task execution and natural language interaction. Models in the 7 to 30 billion parameter range. GPU-favorable when the model fits in available GPU memory; falls back to CPU or escalates to cloud otherwise. Invocation pattern is request-driven, typically below 0.1 Hz per agent.

We parameterize the workload mix as $(\alpha_T, \alpha_S, \alpha_L)$ where $\alpha_T + \alpha_S + \alpha_L = 1$ represents the fraction of agents in each tier. Three reference mixes capture the range of plausible deployments:

Mix	α_T	α_S	α_L
Sensing-heavy (sensing dominates, meaningful reasoning)	0.50	0.35	0.15
Balanced (mixed reasoning across all three tiers)	0.35	0.40	0.25
Reasoning-heavy (active LLM use, lighter sensing)	0.20	0.35	0.45

These mixes are illustrative reference points spanning the plausible space of agentic deployments. They are not claims about market distribution; real deployments will fall somewhere within or between these reference points. A physical AI deployment with multiple sensor agents, a planner, and a language interaction agent would fall in the Balanced region. A continuous health monitoring deployment with many TinyML sensor classifiers and a lightweight coordinator would fall toward Sensing-heavy. A task-execution agent with conversational interface and tool use would fall toward Reasoning-heavy. The structural conclusions of the analysis depend on where in the parameter space a workload falls, not on the application domain that produced it.

In addition to per-tier inference work, the agentic stack incurs operational overhead from agent runtime lifecycle, service discovery, inter-agent messaging, authentication, and routing. We parameterize this overhead through a operational ratio ρ, defined as the fraction of total CPU time spent on operational work rather than on inference itself. We sweep ρ across the range $[0.05, 0.50]$. The lower bound corresponds to minimal operational overhead (agents operating largely independently); the upper bound corresponds to heavy operational load (frequent inter-agent messaging on small inferences). This range is consistent with the inter-agent communication overhead characterized in [5] and the CPU and network call density observed in measured agentic workflows discussed in Section 7.

Realistic deployments typically operate in the lower half of this range. Single-device multi-agent stacks with moderate inter-agent dependence (the common case for physical AI, on-device assistants, and ambient health monitoring) usually sit at $\rho = 0.05$ to $\rho = 0.20$. Higher values of ρ correspond to specific deployment patterns: high-frequency tool-using agents that make many small external calls per inference, federated stacks where agents are distributed across multiple devices on a local network, or deployments dominated by many low-cost inferences with proportionally large routing overhead. Values of ρ above 0.30 are best read as stress test conditions or pathological operating points rather than typical operation. We retain the full sweep up to $\rho = 0.50$ to characterize platform behavior under operational stress; figure captions identify the representative ρ for each chart.

The operational ratio ρ defined above is a lumped parameter. The underlying CPU work splits into a structural floor that any distributed agentic system pays (inter-agent communication, zero-trust security, and the irreducible portion of service discovery) and an optimizable remainder (lifecycle, observability, routing logic, pre- and post-processing, policy enforcement). The structural floor consumes exclusively CPU; no accelerator can offload protocol parsing, signature verification, or registry lookup at any practical scale. Empirical anchoring comes from the MeshInsight study of service mesh sidecar overhead [9], which measured CPU usage

increases of 42 to 163 percent from service mesh deployment in production microservice applications. A full decomposition of ρ into its components, with per-component cost ranges and per-call cost equations, is presented in Appendix A.

4 Analytical Model

The agentic workload on a heterogeneous SoC is modeled as the concurrent execution of inference work distributed across three compute resources (CPU, GPU, NPU) plus operational work on the CPU. We compute resource utilization, energy consumption, and memory footprint as functions of the workload parameters and the silicon platform.

4.1 Per-Tier Inference Work

For an agent in tier $i \in \{T, S, L\}$ with model size m_i and invocation rate f_i, the inference work per second is:

$$W_i = f_i \cdot c_i(m_i, P) \tag{1}$$

where $c_i(m_i, P)$ is the per-inference compute cost on platform P, in units of CPU-seconds, GPU-seconds, or NPU-seconds depending on which resource the tier is placed on. The cost c_i is computed from the per-operator profiling reported in [1] and the per-model throughput measurements reported in [2, 3].

We define operator-affinity weights $w_i^{CPU}, w_i^{GPU}, w_i^{NPU}$ for each tier, with $w_i^{CPU} + w_i^{GPU} + w_i^{NPU} = 1$, representing the fraction of inference work that executes on each compute resource. For the three tiers we use the following weights, drawn from the literature and the empirical observations in [2] and [4]:

Tier	w^{CPU}	w^{GPU}	w^{NPU}
TinyML	0.20	0.05	0.75
sLLM (small models, prefill-dominated)	0.60	0.10	0.30
LLM (larger models, decode-dominated)	0.20	0.65	0.15

These weights vary with model size and silicon affinity; the values above represent the default operating point and are subject to sensitivity analysis in Section 6.

4.2 Operational Work

The CPU carries the agentic system's operational plane: the data management (serialization, context and state), transport and I/O, inter-agent collaboration, security, discovery, routing, and lifecycle work that surrounds inference. We refer to this collectively as operational work. None of it can be offloaded to a GPU or NPU. Operational work is CPU-only and scales with both agent count and inter-agent message rate. We model operational CPU demand as:

$$W_{coord} = \rho \cdot \frac{W_{inference}^{total}}{1 - \rho} \tag{2}$$

where $W_{inference}^{total} = \sum_i N\alpha_i W_i$ is the total inference work across all agents. This formulation captures the operational ratio ρ as the fraction of total CPU work that is operational rather than inference, while keeping the relationship algebraically consistent: $\rho = W_{coord}/(W_{coord} + W_{inference,CPU})$ where $W_{inference,CPU}$ is the CPU-bound portion of inference.

The operational cost includes service discovery, authentication and zero-trust messaging, message routing, agent lifecycle (cold-start and keepalive management), and event handling. Per-call overhead ranges drawn from [5] are used to validate that the modeled ρ values are consistent with the underlying primitive costs.

A per-call operational cost model that combines payload-dependent (protocol parsing, encryption) and per-call fixed (network stack, mTLS, token validation, signature verification, discovery) terms is developed in Appendix A. For a representative zero-trust agentic call with a 2 KB payload, mTLS, JWT validation, and signed messages, the per-endpoint CPU cost evaluates to approximately 200 μs on the X100's 16-core Zen 5 complex and roughly two to three times that on the GPU-centric class's 14-core Neoverse-V3AE complex. This per-call ratio compounds with the aggregate CPU throughput gap between the two complexes to produce a severalfold difference in the capacity fraction consumed at any given inter-agent call rate.

4.3 Resource Utilization

For each compute resource $R \in \{CPU, GPU, NPU\}$, total demand per second is:

$$D^{CPU} = W_{coord} + \sum_i N\alpha_i f_i c_i w_i^{CPU} \tag{3}$$

$$D^{GPU} = \sum_i N\alpha_i f_i c_i w_i^{GPU} \tag{4}$$

$$D^{NPU} = \sum_i N\alpha_i f_i c_i w_i^{NPU} \tag{5}$$

Utilization on each resource is the ratio of demand to capacity:

$$U^R = \min\left(\frac{D^R}{C^R(P)}, 1\right) \tag{6}$$

where $C^R(P)$ is the capacity of resource R on platform P, expressed in the same units as D^R. CPU capacity is the product of core count and per-core throughput, with parallel scaling efficiency factor η applied for $N > 1$ cores. GPU and NPU capacities are taken from published peak throughput at the precision corresponding to the dominant inference workload.

4.4 Energy

Steady-state power consumption is computed as the sum of per-resource utilization-weighted power draw:

$$P_{avg} = P_{CPU} \cdot U^{CPU} + P_{GPU} \cdot U^{GPU} + P_{NPU} \cdot U^{NPU} \tag{7}$$

where P_R is the peak power draw of resource R and U^R is its utilization. We assume idle power is amortized across the cycle and that resource utilization scales power approximately linearly.

4.5 Memory Footprint

Total memory footprint is the sum of agent state, model weights (deduplicated across agents sharing the same backbone), and runtime overhead:

$$M_{total} = M_{runtime} + N \cdot M_{agent} + \sum_i (1 - \delta_i) m_i^{shared} \tag{8}$$

where $M_{runtime}$ is the agent runtime baseline footprint, M_{agent} is the per-agent state, m_i^{shared} is the shared model weight footprint in tier i, and δ_i is the deduplication factor (drawn from [5], where 59 percent storage savings via deduplication was demonstrated).

5 Reference Platforms

We instantiate the analytical model on two heterogeneous SoCs, one from each architectural class. All parameters are drawn from publicly available silicon documentation.

5.1 CPU-Centric Class: AMD Ryzen AI Embedded X100

The X100 series, based on the Strix Halo architecture, combines Zen 5 CPU cores, an RDNA 3.5 integrated GPU, and an XDNA 2 NPU on a single SoC with unified memory [7]. As full X100 specifications were not public at the time of writing, the detailed parameters used here are drawn from the Ryzen AI Max+ 395 (Strix Halo) public specification as the closest available proxy; the NPU is held at the conservative 50 TOPS figure of the P100, which AMD states the X100 exceeds, so the X100 is if anything understated in this analysis.

Parameter	Value
CPU	16 Zen 5 cores, up to 5.1 GHz, 80 MB cache
GPU	RDNA 3.5, 40 CUs, integrated
NPU	XDNA 2, 50+ TOPS (INT8)
Memory	Up to 128 GB LPDDR5X, 256-bit, ∼218 GB/s, unified
Power envelope	45 to 120 W (configurable)

5.2 GPU-Centric Class

The GPU-centric class is characterized by a dominant GPU with Tensor accelerators, supported by a moderate Arm CPU complex and unified memory, with no separate NPU. Parameters are drawn from the NVIDIA Jetson AGX Thor (T5000) module as a representative production member of this class [8].

Parameter	Value
CPU	14 Arm Neoverse-V3AE cores, up to 2.6 GHz
GPU	Blackwell class, 2560 CUDA cores, 96 Tensor cores, up to 2070 FP4 TFLOPS
Memory	128 GB LPDDR5X, 273 GB/s, unified
Power envelope	40 to 130 W (configurable)

5.3 Normalization for Comparison

CPU, GPU, and NPU contributions are measured in different native units (normalized CPU throughput, FP16 TFLOPS, INT8 TOPS) and must be normalized to a common scale for the class definitions in Section 3. We use the following normalization, applied consistently across both platforms:

- CPU normalized throughput converted to INT8-equivalent TOPS using a conversion factor derived from per-operator measurements in [2] for the small-model regime where on-device CPU inference is most competitive.
- GPU FP16 TFLOPS converted to INT8-equivalent TOPS at a 2x ratio, reflecting that transformer-class kernels achieve approximately 2x INT8 over FP16 on these architectures.
- NPU INT8 TOPS used as the reference unit.

The compute distribution shown in Figure 1 reflects this normalization. The exact normalization factors are encoded in the simulation code and can be adjusted; sensitivity analysis shows the qualitative class distinctions hold across a range of plausible normalizations.

6 Simulation Results

We sweep the analytical model across five dimensions:

1. Agent count $N \in \{2, 4, 6, 8, 10, 12, 16, 20\}$
2. Workload mix $(\alpha_T, \alpha_S, \alpha_L)$ across the three reference mixes
3. Operational ratio $\rho \in \{0.05, 0.10, 0.15, 0.20, 0.30, 0.40, 0.50\}$
4. Model size class $\in \{\text{Small}, \text{Medium}, \text{Large}\}$
5. Silicon platform $\in \{\text{X100 (CPU-centric)}, \text{Thor (GPU-centric)}\}$

The full sweep produces $8 \times 3 \times 7 \times 3 \times 2 = 1008$ operating points. The simulation code that produces all results in this section is available for verification, with every parameter labeled with its public source and every equation in Section 4 mapped to a corresponding function.

6.1 Per-Engine Utilization

Figure 2 shows how the agentic workload occupies each compute engine. The columns are the CPU, GPU, and NPU; the rows are the three workload mixes. Each curve is a platform, drawn under single-engine placement, where every model tier runs on the single engine that executes it fastest on that platform. This is a descriptive view of where the work lands across the engines. It is not a ranking of the platforms; the comparison that determines architectural fit is the combined-headroom result in Section 6.2.

The CPU column shows the clearest separation. The X100 holds the lowest CPU utilization in every mix, a consequence of its roughly 2x aggregate CPU throughput over the GPU-centric class, while the GPU-centric class climbs toward saturation as agent count grows, because operational work is CPU-bound and scales with agent count.

The GPU and NPU columns read differently from the CPU column. The GPU-centric class carries a very large GPU and routes essentially all of its inference there, so its GPU column stays low relative to its capacity while its CPU column climbs. The X100 spreads inference across its GPU and NPU: its sLLM tier runs faster on its integrated GPU than on its 50 TOPS NPU, so it routes that tier to the GPU, while TinyML and quantized decode land on the NPU. Because the GPU-centric part has no separate NPU, its quantized inference consolidates on the GPU, which is ample; its binding resource under agentic load is the CPU, not the accelerator.

A single-engine utilization surface cannot, on its own, say which platform is the better fit, because a platform that shows higher utilization on one engine is simply using that engine. Architectural fit depends on the spare capacity left on whichever resource binds, taken across all three engines together. That is the subject of the next section.

6.2 Combined Compute Headroom

Figure 3 shows physical compute headroom: the spare capacity on whichever resource binds each platform, under policy-driven placement in which mimOE distributes each workload across the available engines to minimize the binding-resource utilization. Placement here is a policy decision, not a mechanical handoff: for each inference tier the operating environment weighs the tier's execution cost on each engine, the current headroom on that engine, and escalation and policy constraints before selecting where the work runs. This is the metric that combines all three engines into one comparison, and it is the metric that determines architectural fit.

The X100 retains roughly 80 percent headroom at this operating point in the Sensing-heavy and Balanced mixes and does not saturate anywhere in the modeled space. The GPU-centric class falls toward saturation as agent count grows, bound by its CPU. Across the 455 feasible

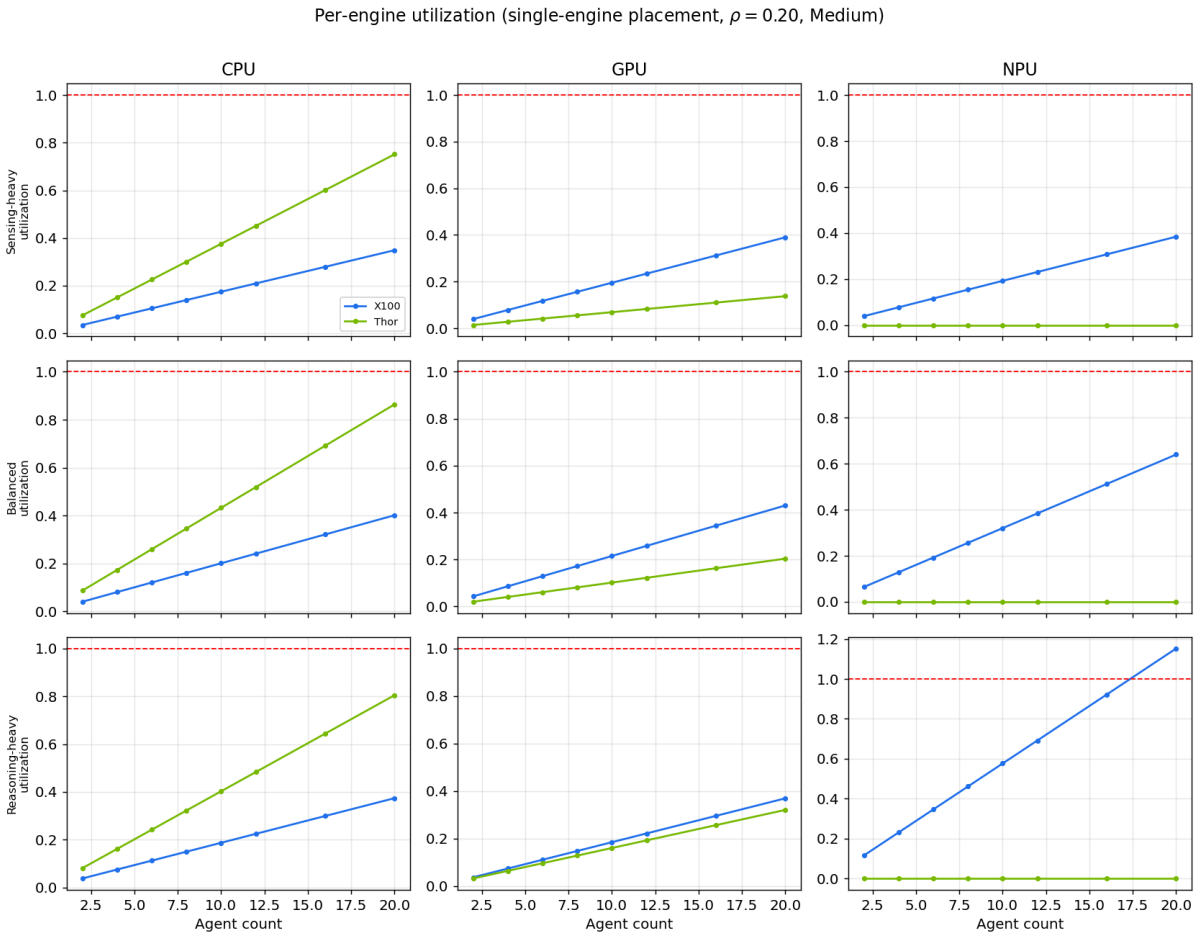

Figure 2: Per-engine utilization under single-engine placement (columns: CPU, GPU, NPU; rows: workload mix), at $\rho = 0.20$, Medium model size. Each tier runs on the engine that executes it fastest on each platform. These are utilization surfaces, not headroom rankings; the combined-headroom comparison is Figure 3.

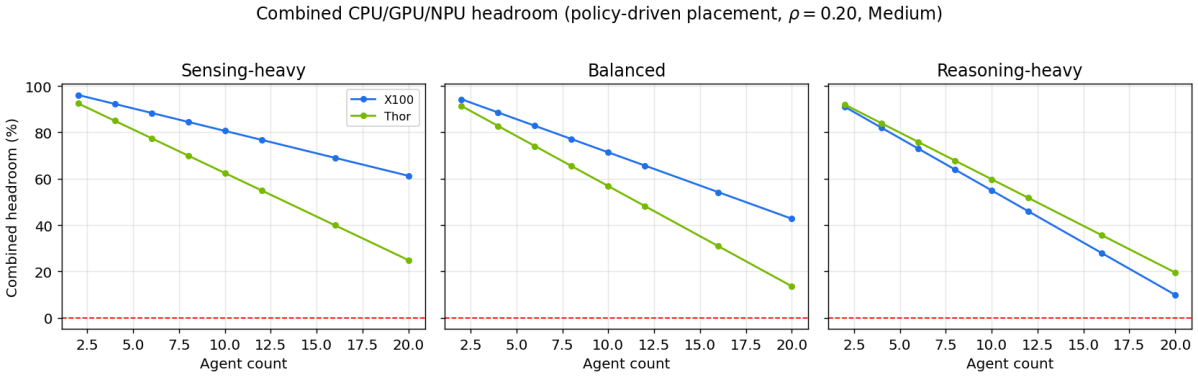

Figure 3: Combined CPU/GPU/NPU headroom (spare capacity on the binding resource) under the dynamic load-balancing policy, per workload mix, at $\rho = 0.20$, Medium model size. The zero line is saturation.

scenarios in the parameter sweep, the X100 holds the most combined headroom in 82 percent of cases under this policy, winning all Sensing-heavy and Balanced workloads and ceding 57 percent of Reasoning-heavy, LLM-dominated workloads to the GPU-centric class, whose larger GPU carries that work with more headroom. Under static single-engine placement the X100

holds the most in 74 percent. The two regimes bracket realistic runtime behavior; averaged across the full sweep the X100 retains about 70 percent headroom on its binding resource against 60 percent for the GPU-centric class.

The mechanism is that agentic workloads are operationally bound: the operational plane runs on the CPU and cannot be offloaded to an accelerator. A platform's combined headroom is set by its weakest needed resource. For the GPU-centric class that resource is the CPU, which binds under operational load in every workload except the ones where a single large model dominates and its large GPU becomes the deciding resource. The CPU-centric X100 has no comparably weak resource across the operationally intensive range, which is why it retains the most spare capacity across the broadest span of workloads, while the GPU-centric class leads precisely in the large-model reasoning region its GPU is built for.

6.3 CPU Utilization Across the Operational Plane

Figure 4 shows CPU utilization on each platform across the (agent count, operational ratio) plane for the Balanced workload mix at Medium model size. The X100 maintains low CPU utilization across the parameter region considered, with peak values around 0.25 at the highest agent counts and operational ratios. The GPU-centric class reaches 0.85 or higher in the same region.

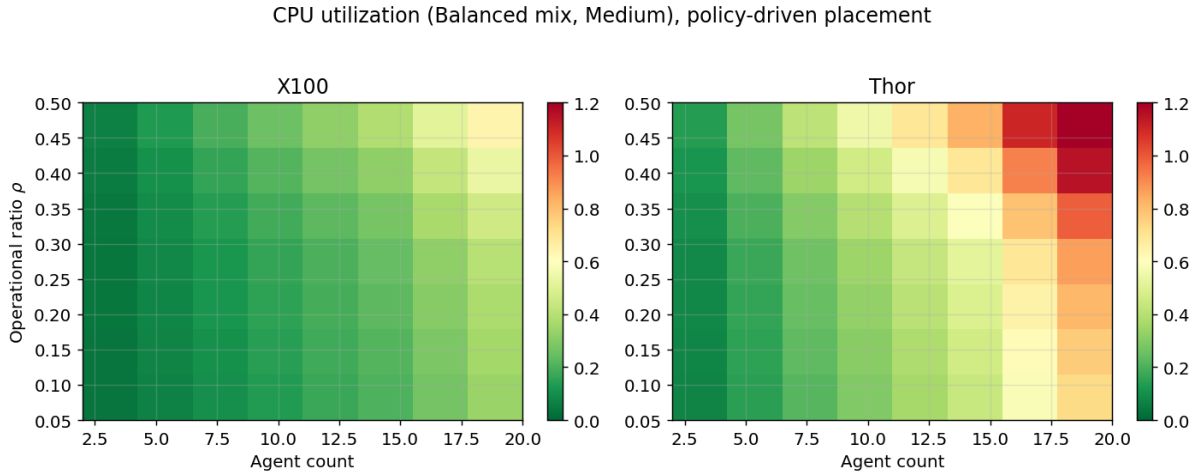

Figure 4: CPU utilization across the two architectural classes for the Balanced workload mix at Medium model size, across agent count and operational ratio. Values above 1.0 indicate oversubscription, where the workload demand exceeds the platform's CPU capacity.

6.4 GPU Utilization

The two classes exercise their GPUs differently. On the X100 the sLLM tier runs faster on the integrated GPU than on the 50 TOPS NPU, so mimOE places that tier on the GPU, and the GPU column of Figure 2 shows the X100 carrying moderate GPU occupancy. The GPU-centric class carries a far larger GPU and routes essentially all of its inference there; its GPU stays lightly loaded relative to its capacity, and its binding resource under agentic load is the CPU rather than the accelerator.

The case that matters for architectural fit is the reverse: in the Reasoning-heavy mix, where the LLM tier dominates and a single large model runs, the GPU-centric class's very large GPU is the deciding resource and it leads. Across the operationally intensive remainder of the space its CPU binds first. The X100's GPU is neither its weakest resource nor left idle, which is part of why it retains headroom across the full workload range rather than in a narrow region.

6.5 NPU Utilization

The NPU comparison is a nuance in the results. The GPU-centric class has no separate NPU; it consolidates quantized inference on its GPU, which is large enough to carry it. The X100's XDNA 2 NPU, rated at 50 TOPS (a figure AMD states the X100 exceeds), handles continuous TinyML and quantized decode on a dedicated low-power engine, which frees the GPU for the sLLM tier. In Sensing-heavy workloads, where inference placed on the NPU dominates, the X100's NPU is the first of its own engines to load, but it stays well below saturation across the modeled range.

6.6 Saturation Regions

Figure 5 shows the regions of the (agent count, operational ratio) parameter space where CPU saturation occurs on each of the two architectural classes, for Large model workloads where saturation effects are most visible. (At Small model size, no platform saturates anywhere in the explored region; that case is not shown.) Color encodes how many of the two platforms saturate at each operating point: green for zero, yellow for the GPU-centric class only, red for both. Regions colored red are the operating points where single-device deployment is infeasible regardless of architectural class, and cluster scale-out is required. Latency degradation under saturation is not modeled directly in this paper, but is captured indirectly through these saturation regions: any point at which a platform crosses into a saturated regime indicates that the platform's latency under that workload would grow without bound under standard queueing assumptions.

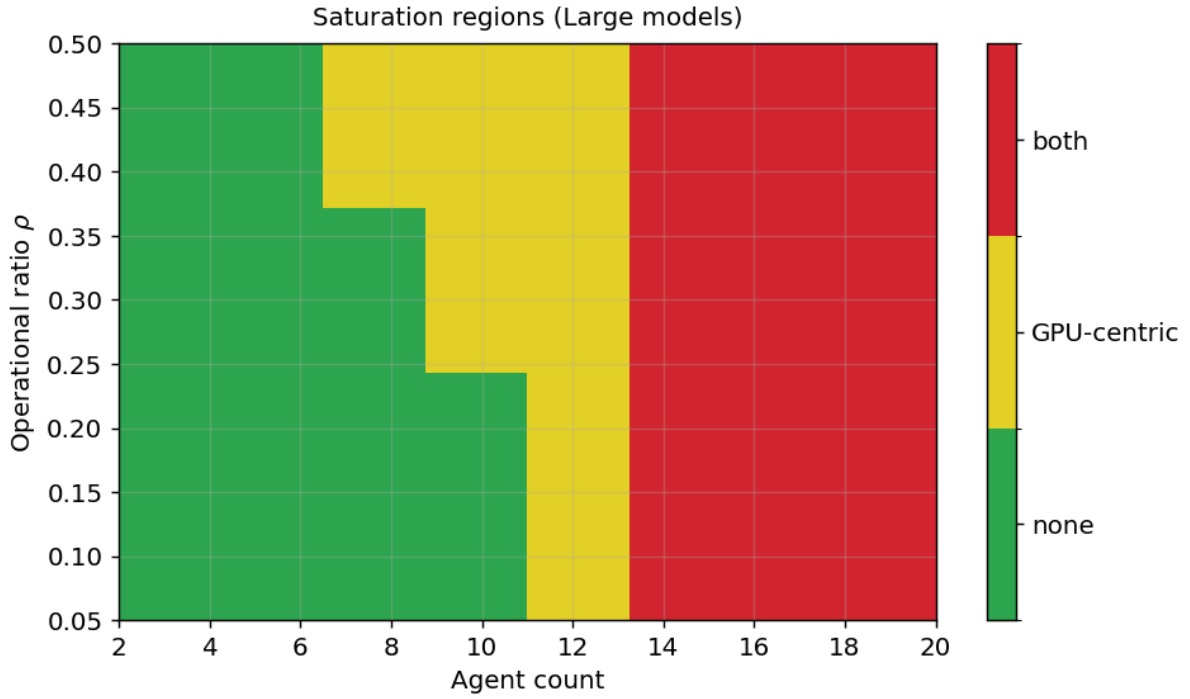

Figure 5: CPU saturation regions across the two architectural classes for Large model workloads. Color encodes which platforms saturate at each (agent count, operational ratio) point: green (none), yellow (GPU-centric only), red (both including X100). The saturation order is consistent across all three workload mixes.

6.7 Energy and Memory

Memory footprint scales with agent count and model size. Figure 6 shows the memory footprint of the agentic stack against the 128 GB capacity that both classes provide. Under microservice deployment the workload fits across the agent-count range at all three model sizes. Under monolithic deployment, where each agent loads its own model copy, the footprint crosses the 128 GB ceiling at high agent counts and Large model size; this is a deployment-model effect rather than a platform-capacity difference between the two classes.

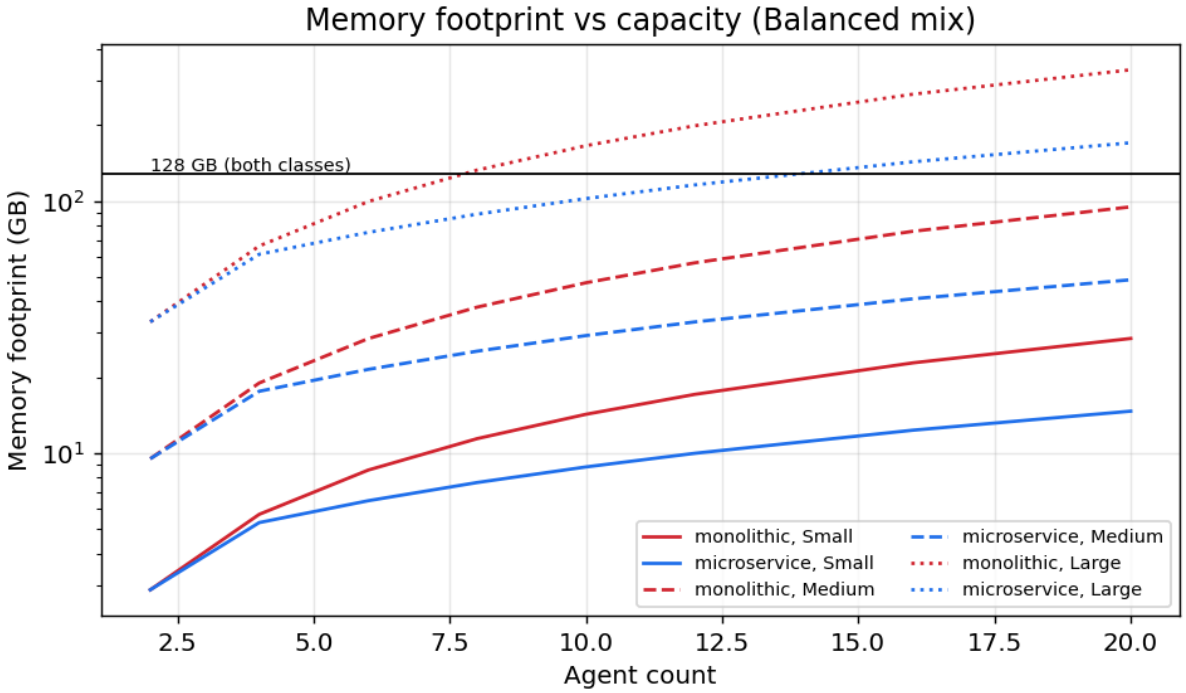

Figure 6: Workload memory footprint vs platform capacity for each model size class (Balanced mix, $\rho = 0.20$).

Energy per second of steady-state operation is close between the two classes. Figure 7 summarizes per-metric results. The X100 has the lowest energy in 51 percent of scenarios and the GPU-centric class in 49 percent. The X100 leads on energy in CPU-bound and high-agent-count workloads, where its CPU efficiency outweighs its higher peak power envelope, while the GPU-centric class leads in low operational overhead, smaller-model workloads where its accelerator carries the inference at lower system power. Energy is therefore a wash between the two classes rather than a differentiator.

6.8 Sensitivity to the Operational Ratio

Figure 8 shows how CPU utilization changes with the operational ratio ρ at fixed agent count and workload mix. The GPU-centric class's CPU utilization grows substantially faster than the X100's as operational overhead increases, reflecting the X100's larger CPU capacity. The X100's curve is also flatter, meaning the X100 is less sensitive to uncertainty in ρ. This robustness is important because ρ is the parameter most sensitive to specific application choices.

7 Measured Workload Examples

To validate that the analytical model covers realistic workloads, we situate measured examples from existing agentic deployments within the modeled space. These examples come from differ-

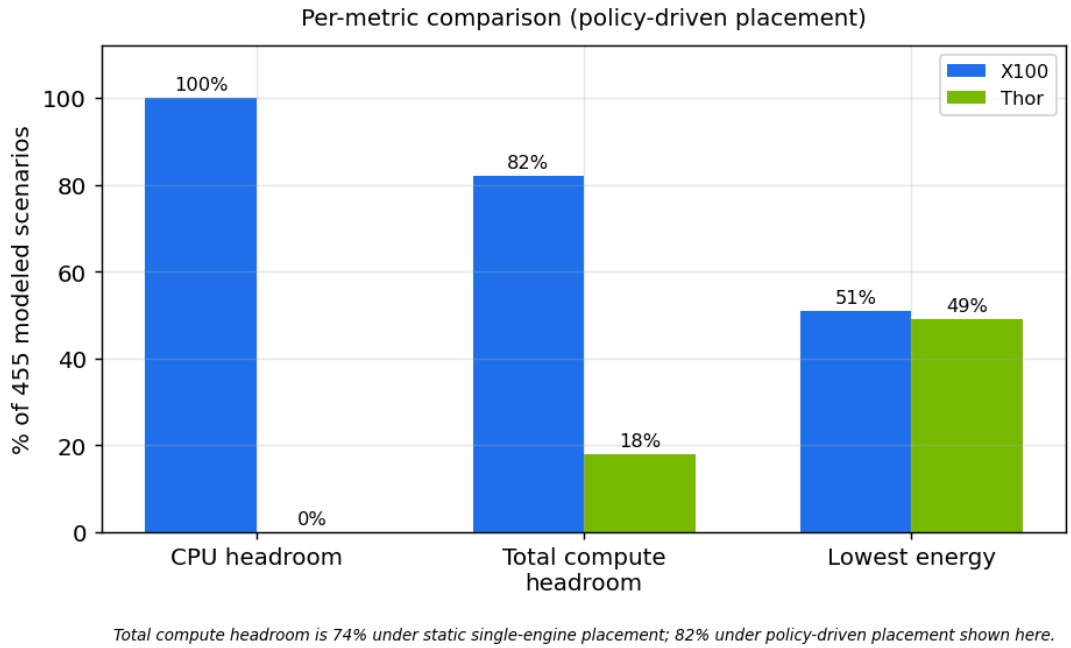

Figure 7: Per-metric comparison across the two architectural classes (455 feasible scenarios). For each metric, count of scenarios in which each platform is strictly most favorable.

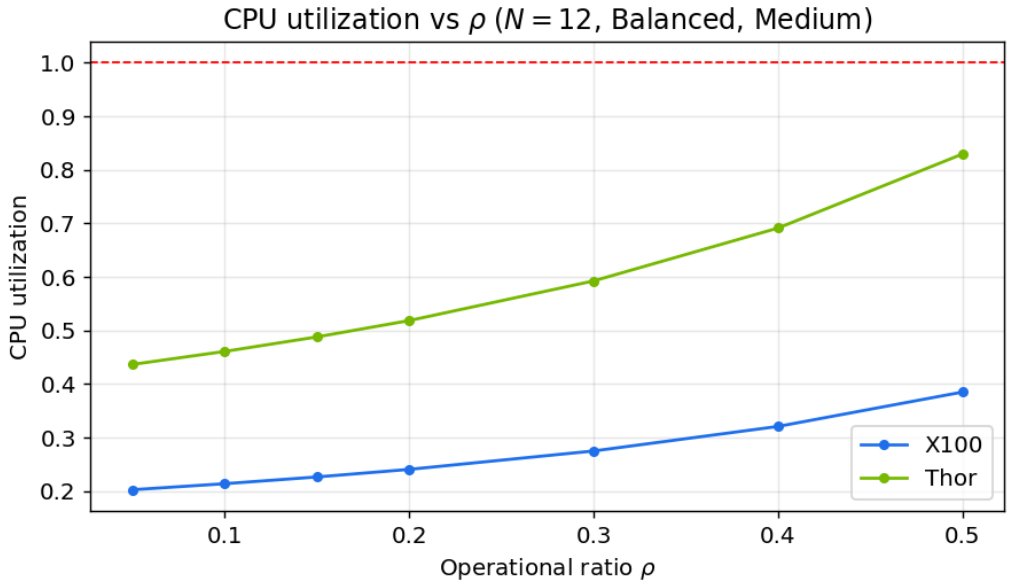

Figure 8: CPU utilization vs operational ratio ρ at $N = 12$, Balanced mix, Medium models.

ent application domains and demonstrate that the parameter space spans the operating points observed in production and pilot deployments.

7.1 Cognifix Video Analysis Workflow

Cognifix is a video analysis product built on mimOE that distributes the analysis of incoming video across multiple coordinating agents. A traced single agentic request in the Cognifix pipeline comprises 28 API calls across 3 nodes and 6 services. Of the 28 calls, 23 (82 percent) are CPU and network bound (coordination, deployment, data transfer, video preprocessing, tracing), and 5 (18 percent) are GPU-bound (vision-language model inference). The workflow

sits at high operational ratio and moderate agent count in the modeled space, with the GPU work concentrated in a narrow phase of the cycle. This example confirms that operationally intensive agentic workloads exist in production and that the modeled ρ range extends to the operating point measured in this trace.

7.2 Intrusion Detection Demo (Physical AI)

A two-agent coordination scenario (camera agent detects an intrusion event, mobile agent moves to engage and notify) deployed on an ARM-based small robotic platform. The workflow sits at low agent count and low operational ratio in the modeled space, with Sensing-heavy mix. This represents a minimal operational overhead operating point at the low end of the agent count axis and demonstrates that the agentic pattern is feasible on constrained end-device hardware.

7.3 Expected Operating Points

Beyond the measured examples above, the modeled space accommodates expected operating points for several Device-First Continuum AI application classes:

- **Complex physical AI platforms** (humanoid robots, autonomous mobile platforms with full perception-planning-language stacks): high agent count (10 to 20), Balanced mix, moderate operational ratio.
- **Ambient health monitoring with multiple physiological signals**: moderate agent count (5 to 10), Sensing-heavy mix, low to moderate operational ratio.
- **Intelligent space deployments with multiple camera and sensor agents**: moderate to high agent count (8 to 15), Sensing-heavy or Balanced mix depending on whether language interaction is included.
- **Conversational task-execution agents with tool use**: low to moderate agent count (3 to 8), Reasoning-heavy mix, moderate operational ratio.

These expected locations are presented to illustrate the range of agentic workloads the framework covers; they are not measured points within this paper and should be confirmed empirically as such deployments mature.

8 Use Case Analysis Across Architectural Classes

The parameter sweep in Section 6 treats agents as homogeneous instances of their tier. Real agentic workloads consist of agents with heterogeneous activity profiles. A perception agent runs continuously while an event-triggered anomaly classifier fires rarely. A coordinator runs on a periodic cadence while a language agent responds to user requests. We analyze four use cases representative of agentic deployment classes to show how the two architectural classes serve realistic agentic workloads.

8.1 Per-Agent Activity Profiles

Each agent in a use case is characterized by an activity profile: continuous (duty cycle near 1.0), periodic (scheduled execution at low duty cycle), event-triggered (fires on upstream events at very low duty cycle), or request-driven (fires on user or external request). Agents that integrate historical records and conversational context pay a per-call cost multiplier of approximately 2 to 4x baseline due to retrieval, context formatting, and reasoning over more tokens.

Duty cycles are drawn from the published surveillance and robotics literature where available. Continuous perception in smart cameras runs at sensor frame rate (duty $\approx$ 1.0) [3].

Event-triggered alerts in deployed surveillance fire at minutes-to-hours intervals [1], yielding duty cycles in the 0.01 to 0.05 range. Robotic action triggered by detection follows the cadence of physical events. Conversational interaction sees active engagement on the order of 5 percent of wall time even for engaged users.

8.2 Use Case Compositions

We model four use cases:

- **Camera anomaly detection with robotic response** (5 agents). One continuous perception agent (TinyML), one periodic supervisor, and three event-triggered agents (anomaly classifier, planner, actuator notifier). Models a security or inspection scenario.
- **Industrial multi-sensor monitoring** (8 agents). Six continuous TinyML classifiers on independent sensor streams, one periodic supervisor, one event-triggered alert agent. Models a factory or infrastructure monitoring deployment.
- **Ambient health monitoring** (8 agents per patient). The workload modeled here is based on Zoara, a preventive health monitoring product built on mimOE. Biometrics are computed on the wearable and sensor devices, which stream high-level biometric summaries to the SoC rather than raw signal. Domain agents (Sleep, Stress, HRV/Recovery, Metabolic, Activity, Clinical/Preventive) reason over those summaries together with historical records and conversational context. A Coordinator agent routes work to the domain agents. A Companion sLLM, a Llama 3.1 8B model loaded once as a shared service, renders natural language. Because raw-signal processing sits on the sensing devices, the on-SoC work per patient is light reasoning and coordination rather than continuous signal classification. We evaluate two deployment scales: single-patient on smartphone-class hardware, and a multi-tenant ward deployment on workstation-class hardware with heterogeneous patient states (a mix of sleeping, resting, active, and conversing patients, weighted by typical clinical activity patterns).
- **On-device conversational agent** (5 agents). Continuous keyword spotter, periodic context updater, request-driven dialog LLM, event-triggered intent router and tool dispatcher.

8.3 Results

Figure 9 shows the peak resource utilization (maximum across CPU, GPU, NPU) and the memory footprint for each use case on each architectural class.

The results across the four use cases follow distinct patterns:

Light agentic workloads (5–8 agents with event-driven duty cycles) fit easily on both architectural classes. Peak utilization stays below 0.4 across CPU, GPU, and NPU on every platform for the Camera, Industrial, single-patient health monitoring, and Conversational cases. The differences between classes are modest at this scale.

Resource-specific differences emerge in the details. The GPU-centric class shows higher CPU utilization on the context-integration-heavy single-patient health monitoring case (roughly 0.3 CPU versus 0.08 on the X100) because its Arm CPU bears the cost of retrieval and reasoning over historical and conversational context, work that is CPU-bound on both platforms but leaves less headroom on the smaller CPU. On the inference-dominated cases the GPU-centric class's large GPU carries the load comfortably, and at these light agent counts the differences between the two classes are modest.

Multi-tenant scale exposes structural differences. With biometrics computed on the sensing devices, the per-patient on-SoC load is light reasoning over summaries plus coordination, and the deployment model is what determines whether the workload fits. Under a monolithic stack each patient process loads its own copy of every model, dominated by the companion and

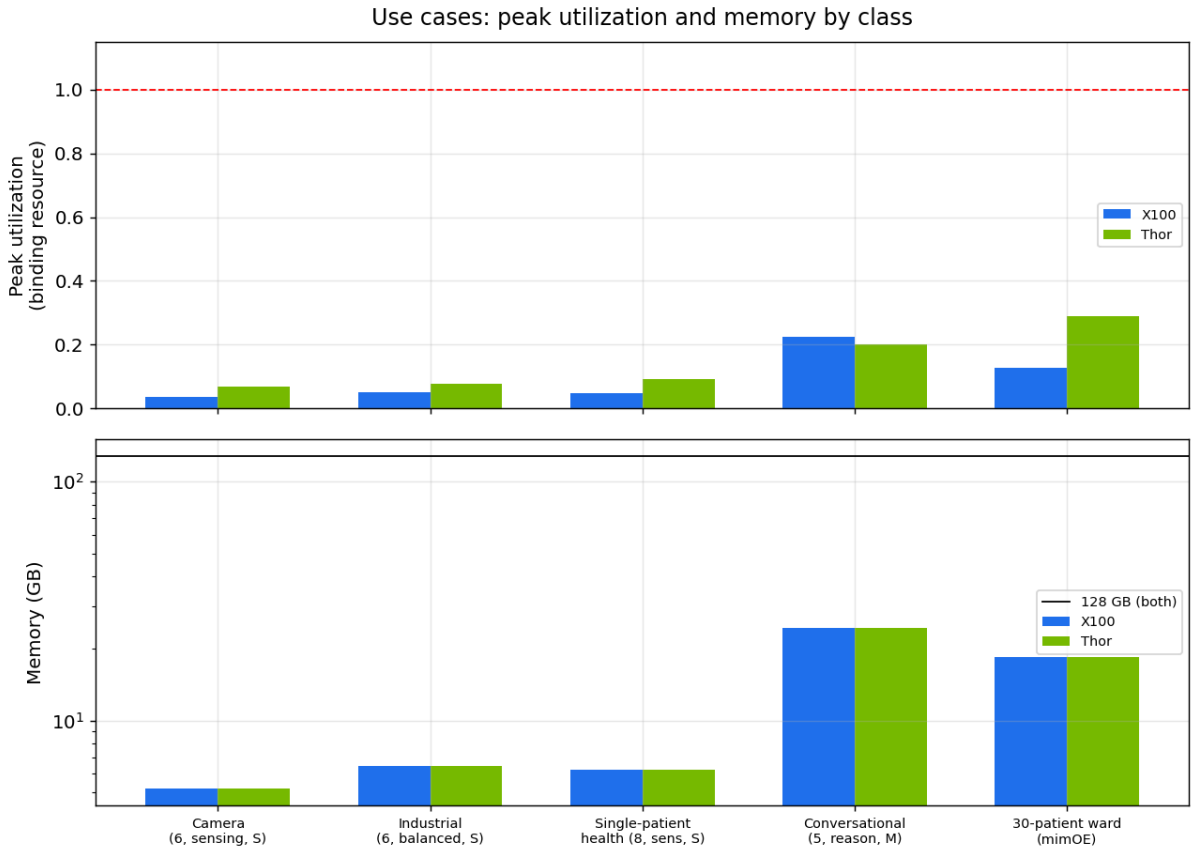

Figure 9: Peak resource utilization (top) and memory footprint (bottom) for each use case on each architectural class. The multi-tenant health case is shown as a 30-patient ward under mimOE, where the workload is bound by operational work on the CPU and its footprint fits in memory on every class.

coordinator sLLM, at roughly 9 GB per patient. That stack crosses the 128 GB memory ceiling shared by both classes at about 14 patients, so a monolithic deployment is memory-bound before a single ward is served. Under mimOE the models load once as a shared base of roughly 10 GB and each patient adds only about 300 MB of history, state, and conversation context, so a full 30-patient ward occupies under 20 GB and, even at the CPU-bound capacity limit of 234 patients, resident memory reaches only about 85 GB; memory therefore never becomes the binding constraint. Once memory is removed as the limit, the binding resource is operational work on the CPU, since every patient adds a coordinator and a set of domain agents whose orchestration is CPU-bound. On this axis the classes separate by CPU capacity: the X100 sustains on the order of 234 patients per device before the operational plane saturates, against approximately 104 for the GPU-centric class, a 2.25x advantage that tracks the X100's aggregate CPU throughput (Figure 10). At a realistic ward scale of 30 patients both classes hold ample free memory, but the X100 runs the workload below a quarter of every resource (CPU near 0.15) while the GPU-centric class runs its CPU roughly twice as high, leaving less headroom to absorb capability upgrades such as a larger companion model, more domain agents, or longer history windows.

8.4 General Observations on Architectural Fit

Before stating the architectural observations, a context note. Production agentic AI deployments do not yet exist at scale. The agentic systems running today are predominantly research prototypes and limited consumer products with single-digit agent counts at modest duty cycles,

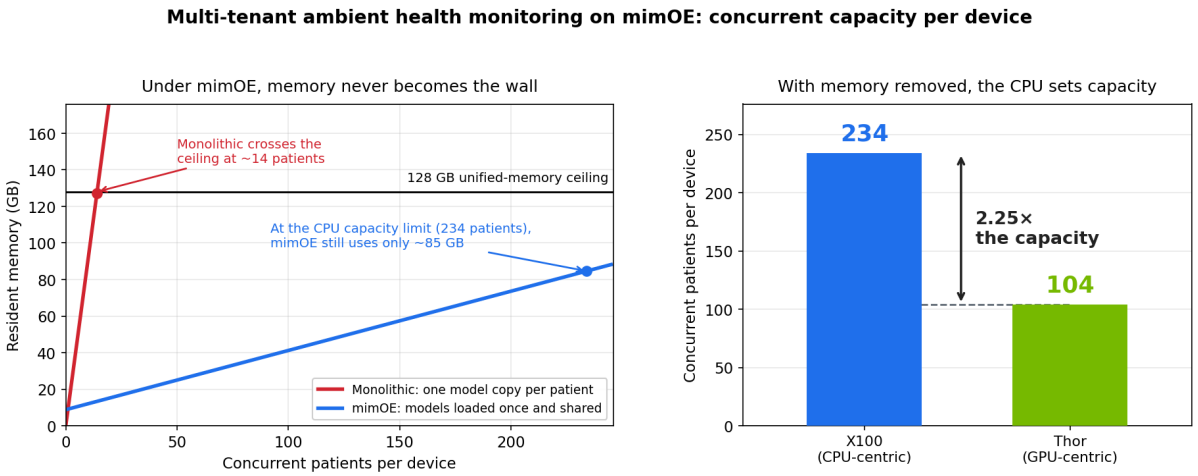

Figure 10: Concurrent capacity per device for the multi-tenant ambient health monitoring deployment on mimOE. **Left:** resident memory versus concurrent patients per device, for a monolithic deployment and for mimOE, against the shared 128 GB unified-memory ceiling. **Right:** CPU-bound concurrent capacity for the two device classes. Values computed by `patient_capacity.py`.

in single-user single-device configurations. Today's deployments fit on any of the three architectural classes with substantial headroom; differentiation only emerges as workloads grow toward what production agentic deployments will look like. The use case results in this section and the parameter sweep in Section 6 project from architectural primitives and published per-operator profiling to the workload shapes future agentic deployments will produce. The observations below describe where each class fits in that projected space, not in the current research-scale regime.

The CPU-centric class covers the broadest range of projected agentic workloads. Across the 455 feasible scenarios in the parameter sweep, the CPU-centric class holds the lowest CPU utilization in 100 percent of cases, because operational work is CPU-bound and cannot be offloaded. On total compute headroom it holds the most in 74 percent of cases under static single-engine placement, and in 82 percent once mimOE balances each workload across the available engines to maximize headroom, winning all sensing-heavy and balanced workloads and ceding only reasoning-heavy, large-model workloads to the GPU-centric class. Averaged across the sweep it retains about 70 percent spare capacity on its binding resource against 60 percent for the GPU-centric class, and it never saturates in the modeled space. It also sustains the largest multi-tenant deployment per device: once mimOE removes memory as the constraint, it sustains roughly 234 patients per device against 104 for the GPU-centric class before the operational plane saturates. The advantage is structural: the CPU-centric class's roughly 2x aggregate CPU throughput accommodates the context integration, retrieval, and operational work that agentic stacks introduce. These advantages compound as deployments scale toward production patterns: more agents per device, more context integration per agent, more multi-tenant configurations per device.

The GPU-centric class wins in the region its architecture optimizes for. The GPU-centric class carries a very large GPU (on the order of 2000 FP4 TFLOPS) and is the stronger choice for workloads dominated by a single large reasoning or vision model, where raw accelerator throughput is the binding resource and operational overhead is light. This is the reasoning-heavy, LLM-dominated corner of the space, where it leads the X100. It operates within a narrower agentic envelope than the CPU-centric class, since its CPU binds first as operational work scales with agent count.

For product lines where the workload profile may evolve, the CPU-centric class

is the structurally favored choice. Agentic workloads are evolving along multiple trajectories at once: agent counts per device are growing, per-agent context integration is becoming standard, multi-tenant deployment is emerging as a meaningful operating point. A product line built on a GPU-centric class optimizes for the single-large-model workload and requires re-architecture if the workload shifts toward coordinated multi-agent operation. A product line built on the CPU-centric class absorbs the full trajectory from one SKU. For OEMs and integrators targeting heterogeneous agentic deployments rather than a single narrow market, this translates directly to broader addressable market and lower architecture risk as production workloads evolve.

This connects to the addressable-market dimension of architectural choice: on mimOE, which routes across CPU, GPU, and NPU on either class, a CPU-centric SoC serves more workload classes per SKU than a GPU-centric SoC tuned for single-model inference.

9 Cluster-Scale Deployment

The analysis in Sections 6 and 8 treats each platform as a single device serving an agentic workload. Many real deployments operate as clusters of devices coordinating to serve a workload that exceeds the capacity of any single member. This section extends the framework to cluster-scale deployment, where the architectural fit observations of Section 8.4 compound with statistical capacity gains from resource pooling.

9.1 The Cluster Scenario

Multi-device deployments arise naturally in agentic AI for three reasons. First, agentic workloads grow with the size of the population being served, and single-device capacity caps the population size per device. Second, cluster deployment provides fault tolerance: a failure in one device degrades capacity but does not halt the system. Third, geographic distribution often makes single-device deployment impossible at all: a hospital, factory, or campus covers more physical ground than one device can serve from one location.

We use the multi-tenant health monitoring deployment as the central example throughout this section. The single-device analysis in Section 8 showed that under mimOE one X100 runs a full ward comfortably: at 30 patients it sits near 0.15 CPU utilization with roughly 109 GB of memory free, and its saturation capacity is on the order of 234 patients before operational work becomes the binding resource. A single device therefore covers a ward with large headroom. Clustering at hospital scale is motivated not by per-ward capacity but by three other factors: a medium hospital serves several hundred patients across many wards; fault tolerance requires that a device failure relocate its patients rather than drop them; and pooling independent ward loads across devices lets the fleet sustain higher utilization at the same reliability target. A representative deployment is a fleet of X100s across the hospital, each hosting one or more wards, with mimOE providing failover and load balancing across the fleet.

Two other cluster scenarios are worth noting briefly. A robotic warehouse fleet may deploy 5 to 10 X100s as coordination hubs distributed across the facility, each serving a subset of mobile robots within its physical zone. A smart building or campus deployment may run 15 to 30 X100s on distinct floors or zones, with each device handling local sensing, occupancy, and environmental control while sharing higher-level inference across the cluster. These deployments differ in topology and workload mix from the hospital case but share the same structural properties that the analysis below addresses.

9.2 Statistical Capacity Gain from Pooling

When agents on different devices have independent load patterns, the aggregate load of a cluster has lower relative variability than the load on any single device. The standard result

from queueing theory is that the coefficient of variation of the aggregate scales as $1/\sqrt{n}$ for a cluster of n independent devices, since variance averages out across the cluster.

Let μ denote the average utilization of a device and σ the standard deviation of its instantaneous load. The coefficient of variation is $CV = \sigma/\mu$. A device provisioned for a target saturation probability p (the fraction of time the device exceeds capacity) sustains an average utilization given by:

$$\mu_1^{max} = 1 - z_p \cdot CV$$

where z_p is the normal quantile corresponding to probability p (for example, $z_p = 1.65$ for $p = 0.05$, $z_p = 2.33$ for $p = 0.01$).

A cluster of n devices with independent loads, where work can be routed to any device with spare capacity at the moment of demand, sees its aggregate load fluctuate with coefficient of variation $CV/\sqrt{n}$. The cluster therefore sustains a higher average utilization per device:

$$\mu_n^{max} = 1 - z_p \cdot CV/\sqrt{n}$$

The extra usable utilization per device that the cluster delivers is:

$$\Delta = z_p \cdot CV \cdot (1 - 1/\sqrt{n})$$

For agentic workloads with $CV \approx 0.4$ (a reasonable midpoint for stacks with many independent agents at varied duty cycles) and a 5 percent saturation probability target ($z_p = 1.65$), the sustainable per-device utilization values are:

Cluster size n	Sustainable utilization per device	Gain vs single device
1	34%	baseline
2	53%	+19 pp
3	62%	+28 pp
5	71%	+37 pp
10	79%	+45 pp
20	85%	+51 pp

The cluster does not produce more compute than the sum of its parts; the raw aggregate capacity is exactly n times a single device. What the cluster produces is better utilization of those parts because each device's spare capacity can absorb bursts on its peers. For typical agentic workload variability, a cluster of 5 X100s delivers approximately twice the usable compute of 5 separately operated X100s at the same reliability target.

Applying this to the multi-tenant health monitoring case: a single X100 provisioned for 5 percent saturation tolerance with $CV = 0.4$ sustains approximately 34 percent average utilization in isolation. Running a full ward of 30 patients the X100 sits well below this, near 0.15 CPU, so a single device carries a ward with margin to spare. The pooling gain matters when devices are driven closer to their sustainable limit at hospital scale. A pool of 3 X100s each sustains approximately 62 percent utilization at the same reliability target, and a pool of 10 sustains approximately 79 percent, so a fleet serving many wards delivers substantially more usable capacity at fixed reliability than the same devices operated independently. The same mechanism provides failover: when one device is lost, its ward's agents relocate to peers with spare capacity rather than going dark.

Sensitivity to the modeling assumptions: at $CV = 0.3$ (less variable load) the gains are smaller because there is less variance to average out; at $CV = 0.5$ (more variable load) the gains are larger. At $p = 0.01$ (stricter reliability target) the baseline single-device utilization is lower so the absolute gains are larger but the relative improvements are similar.

9.3 The Substrate Requirement

The pooling math above assumes that work can be dynamically routed to any device with spare capacity at the moment of demand. This requires four runtime primitives, all provided by the microservice substrate described in Section 1: dynamic resource discovery (each device advertises its current capacity), dynamic service discovery (any agent instance can be located on any device), continuum-transparent routing (callers do not need to know which device hosts the callee), and zero-trust security (cross-device calls are authenticated without requiring shared keys at deploy time).

Without these primitives, agents are placed statically at deploy time. Device A hosts one ward, device B hosts another. Bursts on A cannot use spare capacity on B because no mechanism exists to relocate or duplicate agents mid-flight. The cluster's effective capacity in this case is exactly n times the single-device sustainable utilization, with no pooling benefit. The statistical capacity gain table above collapses to a single number: the same 34 percent per device that one X100 alone would sustain. The cluster has n times the raw capacity but no headroom multiplier.

A monolithic stack distributed across multiple devices falls in the static-placement regime. Each device runs its bundled application binary and the binary on device A cannot invoke functions on device B's process without explicit network code written agent by agent. Even with such code, the absence of service discovery means an overloaded agent on A cannot generally be served by a spare instance on B because there is no general mechanism to locate the spare instance at runtime. The pooling benefit is therefore not just a property of running multiple devices; it is a property of mimOE that turns multiple devices into a pool.

9.4 Architectural Fit at Cluster Scale

The single-device fit observations of Section 8.4 compound at cluster scale. Three effects make CPU-centric cluster members particularly favorable.

First, CPU-centric cluster members can absorb any workload regime. When work is routed from a saturating device to a peer, the peer must be able to handle the relocated work. A CPU-centric device can absorb TinyML, sLLM, or LLM work because it has meaningful capacity across all engines and a strong CPU for the operational plane. A GPU-centric device binds on its CPU once operational load rises. A cluster of GPU-centric devices that needs to relocate operationally intensive work has no good destination because every device in the cluster has the same CPU constraint. A cluster of CPU-centric devices does not have this problem.

Second, the placement problem becomes easier. With CPU-centric cluster members, mimOE can place any agent on any device, simplifying the placement decision to "which device has the most current spare capacity." With GPU-centric cluster members, mimOE must steer operationally intensive agents to devices with CPU headroom, which constrains the placement options and can leave GPU capacity idle when the workload is operationally intensive rather than dominated by a single large model.

Third, deployment evolution is more flexible. Agentic workloads evolve as new agents are added, new capabilities replace older ones, and the mix of TinyML/sLLM/LLM shifts over time. A cluster of CPU-centric devices accommodates this evolution by absorbing whatever workload shape the deployment takes. A cluster of GPU-centric devices may need to be re-architected if the workload shifts away from the single-large-model target.

Combined with the single-device addressable-market argument, the complete picture is that on mimOE a balanced SoC serves more workload classes per SKU than alternative silicon classes, and a cluster of balanced SoCs running mimOE delivers both raw capacity scaling (linear in n) and statistical capacity gain (rising with $\sqrt{n}$), across the full workload mix that balanced silicon can absorb. The cluster scenario amplifies the single-device argument rather than replacing it.

9.5 Limitations

The cluster analysis here is qualitative and statistical rather than simulated. Three limitations are worth noting. First, the pooling math assumes loads on different devices are independent. In the multi-tenant health monitoring case this is largely true (patients are independent), but in deployments where agents coordinate heavily across devices (a robot fleet performing coordinated tasks) the loads may be correlated, reducing the pooling benefit. Second, cross-device coordination adds latency and CPU cost that grows with the operational ratio ρ. On a local network with sub-millisecond inter-device latency this cost is small, but deployments with higher-latency network paths would see reduced effective pooling. Third, the analysis assumes homogeneous cluster members. Mixed clusters (different platform classes within the same cluster) introduce placement constraints that the simple pooling model does not capture.

Detailed simulation of cluster behavior, including correlated loads, network coordination costs, and heterogeneous cluster compositions, is reserved for follow-on work.

10 Deployment Lifecycle Dimension

The analysis above models steady-state compute. A second dimension of the silicon and runtime comparison is the deployment lifecycle: how agents and their underlying models are updated, replaced, and managed across a fleet.

In conventional embedded deployments, TinyML models and inference code are compiled and flashed as native binaries, typically as part of the application image or firmware. A model retrain, capability expansion, or bug fix requires a full firmware update or application rebuild and redeployment. At fleet scale, this is operationally costly and introduces regression risk across components that did not need to change.

Microservice-based deployment, as established in [5], decouples the lifecycle of each agent from the application binary. Each agent ships as a microservice (mim) with its own update path. Model weights are stored in a shared registry with deduplication. Updates are deployed over the air per agent without disturbing other components of the stack. In the representative scenario analyzed by the cited work, microservice-based deployment showed substantial reductions in update payload size, storage requirements through deduplication, regulatory verification scope per agent, and fault propagation radius compared with monolithic deployment. Specific magnitudes are reported in the cited work and depend on workload composition, agent count, and operational parameters; the architectural advantage holds across the range of agentic deployments considered there.

This operational dimension does not change the steady-state compute analysis but is decisive in production deployment at scale, and it is where microservice-based agent runtimes differentiate structurally from native compiled deployment regardless of the underlying silicon.

11 Discussion

The simulation results support the central thesis that agentic AI workloads exercise CPU substantially and that platforms with substantial CPU headroom are structurally favored. Comparing the two architectural classes across the 455 feasible scenarios, the X100 holds the lowest CPU utilization (most headroom under the same workload) in 100 percent of scenarios, because operational work is CPU-bound and the X100's CPU throughput is roughly twice that of the GPU-centric class. Total compute headroom depends on how inference is placed. Under static single-engine placement the X100 holds the most total headroom in 74 percent of scenarios, and under the headroom-balancing placement policy in 82 percent, winning all sensing-heavy and balanced workloads and ceding 57 percent of reasoning-heavy, LLM-dominated workloads to the GPU-centric class, whose larger GPU carries that single-large-model work with more headroom.

On energy the two classes are close, with the X100 lowest in 51 percent of scenarios and the GPU-centric class in 49 percent; the X100's energy standing is strongest in CPU-bound, high-agent-count workloads. Both classes provide 128 GB of unified memory, so memory capacity is not a differentiator between them; its role is discussed in Section 6.7 and at multi-tenant scale in the use case analysis. The result also identifies the operating region where the GPU-centric class is structurally favored, which we discuss below.

11.1 Where the X100 Leads

The X100 leads most strongly in the operating region characterized by high agent count, moderate to high operational ratio, and Medium to Large model sizes. This region corresponds to agentic stacks with substantial coordination among many concurrent agents, which arise in complex physical AI platforms (humanoid robots, autonomous mobile platforms with full perception-planning-language stacks), multi-sensor ambient deployments, and conversational task-execution systems with tool use. In these workloads multiple coordinating agents operate concurrently and exchange messages frequently.

Two structural factors drive the X100's lead in this region. First, the 16-core Zen 5 CPU complex provides roughly twice the aggregate CPU throughput of the 14-core Neoverse-V3AE CPU in the GPU-centric class, with strong single-thread performance. This headroom absorbs the operational overhead that scales with agent count in agentic stacks. Second, mimOE spreads inference across the X100's GPU and NPU, so the CPU headroom is preserved for operational work rather than being spent on inference the accelerators could carry.

11.2 Where the GPU-Centric Class Leads

For workloads dominated by a single large reasoning or vision model with light operational overhead, the GPU-centric class's very large GPU (on the order of 2000 FP4 TFLOPS) far exceeds the X100's integrated GPU. In this reasoning-heavy region, where the large model is the binding resource, the GPU-centric class maintains more headroom and leads the X100.

The GPU-centric class leads on energy in roughly half of scenarios, primarily those characterized by lower agent counts, lower operational ratios, and inference-dominated workloads where its accelerator carries the load at low system power. On energy the two classes are close overall.

11.3 Workload Selection Guidance

The results suggest a workload-dependent selection criterion rather than a universal recommendation:

- **Choose the CPU-centric class (X100) for:** multi-agent stacks with substantial operational overhead; complex physical AI platforms with layered TinyML-sLLM-LLM workloads; multi-tenant deployments; applications where CPU headroom is required to absorb application growth without re-architecting.
- **Choose the GPU-centric class for:** single large-model inference workloads with low operational requirements; vision-heavy and language-model-serving applications where raw GPU throughput is the binding resource and multi-agent coordination is not exercised at scale.

The relative advantage of the CPU-centric class grows with the agentic intensity of the workload. Stacks that resemble single-task inference platforms see modest differences between the two classes, and the largest single models favor the GPU-centric class. Stacks that resemble coordinated multi-agent systems with mixed TinyML, sLLM, and LLM tiers see larger differences, with the direction favoring the CPU-centric class whose CPU absorbs the operational load.

11.4 Combined Story with Microservice Runtime

The compute analysis above models steady-state operation. The deployment lifecycle dimension discussed in Section 10 adds an operational dimension not captured by the steady-state silicon comparison. The microservice-based agent runtime decouples model and capability lifecycle from the application binary, which addresses an operational pain point that scales with fleet size and update frequency. This advantage is independent of the silicon choice in a mathematical sense, but it compounds with the silicon comparison: the CPU-centric class's CPU headroom provides room to run the microservice runtime overhead alongside the agentic workload, while a class with a smaller CPU share leaves less headroom for both.

The combined effect is that mimOE addresses the fleet-scale operational envelope on both classes, while the CPU-centric silicon additionally addresses the steady-state compute envelope of agentic deployments. Neither dimension alone fully characterizes the deployment problem, and the silicon advantage is most clearly realized in the workload regions where the advantage of mimOE also applies.

The structural decomposition of ρ from Section 3 makes the silicon-versus-runtime distinction precise. The structural CPU floor $\rho_{net} + \rho_{sec} + \rho_{disc}$ is independent of runtime efficiency. It is set by the workload class, scales with inter-agent call rate, and consumes only CPU. Silicon choice is the primary lever against this floor: silicon classes with more CPU headroom absorb it as background noise, while silicon classes with less CPU headroom pay it as a measurable fraction of capacity. The optimizable remainder ρ_{rt} is the component over which runtime engineering has direct leverage, through batched lifecycle operations, aggressive caching of discovery results, and connection reuse. The two levers compose. A CPU-centric SoC paired with a well-engineered agentic operating environment addresses both components; a GPU-centric SoC paired with a sophisticated one cannot reduce the structural floor regardless of how well that environment is engineered.

11.5 Future Work

This paper presents a steady-state analysis at moderate agent counts (2 to 20). Two extensions are reserved for follow-on work. First, the scaling behavior to high agent counts (50 to 500), where the comparison between monolithic and microservice deployment is no longer about per-inference efficiency but about whether the architecture fits on the device at all. The Agentix-Native paper's memory measurements suggest monolithic stacks become infeasible well before microservices do, but a complete scaling analysis with the silicon dimension layered on top remains future work. Second, the architectural transition from many specialized TinyML models to fewer general sLLMs at high capability counts, which interacts with silicon class in interesting ways: the CPU-centric class accommodates both regimes well, while a GPU-centric class fits one regime well and the other poorly.

11.6 Limitations and Sensitivity

Four limitations of the analytical model should be noted. First, the operator-affinity weights $(w_i^{CPU}, w_i^{GPU}, w_i^{NPU})$ are modeling choices informed by the published literature but not measured directly on the two platforms. Sensitivity analysis shows the qualitative conclusion is robust to weight perturbations within ± 0.15, but quantitative claims would benefit from platform-specific measurements. Second, the linear power model approximates real silicon power curves that are convex; energy numbers should be read as comparative rather than absolute.

Third, the multi-tenant analysis (the ward-scale health monitoring case in Section 8) models patient states as a weighted mix of sleeping, resting, active, and conversing, with state-specific per-agent multipliers. The aggregate load is therefore an expected value across this distribution rather than a worst-case peak. The state weights and per-state multipliers are modeling choices

informed by typical clinical activity patterns; deployments with different patient-state distributions would shift the aggregate load proportionally. The qualitative finding (under mimOE the workload is bound by operational work on the CPU, and the CPU-centric class sustains more than twice as many patients per device as the GPU-centric class) is robust across the range of plausible state distributions, since the per-patient load is dominated by operational work that is resident regardless of activity.

Fourth, results are reported under two placement regimes that bracket realistic runtime behavior. Under *static placement*, each tier executes on the single resource indicated by its operator-affinity weights, representing a runtime that pins each model to one engine. Under the *headroom-balancing policy*, which a microservice runtime with dynamic resource discovery can implement, each tier's accelerator-eligible work is distributed across the GPU and the NPU to minimize the binding resource utilization; the optimal split is computed exactly at each operating point as a small linear program. Operational work remains CPU-bound and is not offloadable under either regime. Under static placement the CPU-centric class holds the most total headroom in 74 percent of scenarios; under policy-driven placement it holds the most in 82 percent, ceding the reasoning-heavy, large-model region to the GPU-centric class whose GPU is the deciding resource there. The headroom-balancing figure is an upper bound, since real load balancing carries scheduling overhead and granularity limits, so deployed behavior sits between the two regimes. A separate form of placement, relocating agents across devices in a cluster, is treated in the cluster analysis of Section 9 and is not folded into the single-device sweep. The reasoning-heavy result under the policy is the most sensitive to the CPU normalization between platform classes; the sensing-heavy and balanced results rest on the larger and more robust CPU-throughput gap.

The simulation code is structured to make every assumption explicit and changeable, supporting independent verification.

12 Conclusion

We presented a model of agentic workloads on heterogeneous SoCs across five parameters and two architectural classes (CPU-centric and GPU-centric), with the AMD Ryzen AI Embedded X100 as the CPU-centric exemplar and an NVIDIA Jetson Thor-class part as the GPU-centric exemplar. The model is grounded in public silicon specifications and published per-operator profiling, and the simulation code is available so that every parameter and equation can be inspected and re-run independently.

Architectural class matters little for small to moderate workloads, where both classes serve the load with ample headroom. The differences emerge under load. The operational plane is CPU-bound and cannot be offloaded to an accelerator, so the CPU-centric class leads CPU headroom in every scenario, and leads total compute headroom in 74 to 82 percent of scenarios depending on placement, ceding only the large-model reasoning region to the GPU-centric class whose GPU is built for it. At multi-tenant scale the CPU-centric class's larger CPU sustains roughly 2.25 times the concurrent load per device. Energy is comparable between the two classes.

These results assume mimOE on both classes, and mimOE is essential to reach them. In a multi-agent deployment it loads each model once and references it across agents and tenants, pools burst capacity across devices in a cluster, places work across the CPU, GPU, and NPU under policy, and provides service discovery, zero-trust security, fault isolation, per-agent updates, and per-agent observability as architectural defaults rather than per-application engineering. Under a monolithic runtime the 15-patient health workload requires about 136 GB and exceeds the 128 GB ceiling of even the largest current part, so neither class fits a production multi-agent workload without it. mimOE is the common substrate that makes the silicon usable for agentic AI, not an advantage unique to either part, and both classes benefit from it.

On that substrate the silicon choice determines how much capacity the runtime can deliver. The CPU-centric class is the choice that hedges against workload uncertainty: the GPU-centric class leads within the single-large-model regime it is optimized for and falls behind outside it, while the CPU-centric class serves a broader range of agentic workloads per SKU, because the operational plane that dominates agentic work is CPU-bound. This benefits OEMs and integrators targeting heterogeneous agentic deployments. At cluster scale the advantage compounds through statistical pooling across devices, which mimOE provides on both classes. Future work includes calibrating the operator-affinity weights against platform-specific measurements, modeling thermal throttling under sustained load, and extending the framework to additional platforms.

Appendix A Detailed Decomposition of the Operational Ratio ρ

This appendix presents the full decomposition of ρ summarized in Section 3.

A.1 Structural Floor and Optimizable Remainder

The operational ratio decomposes as:

$$\rho = \rho_{net} + \rho_{sec} + \rho_{disc} + \rho_{rt} \tag{9}$$

where ρ_{net} is the CPU cost of inter-agent communication (serialization, network stack traversal, deserialization), ρ_{sec} is the cost of zero-trust security operations (mutual authentication, payload encryption, message signing and verification), ρ_{disc} is the irreducible portion of agent and resource discovery (registry maintenance, cache miss recovery, capability matching for novel requests), and ρ_{rt} absorbs the optimizable remainder (lifecycle management, batched routing logic, observability and distributed tracing, distributed state management, pre- and post-processing, and policy enforcement). The first three components form a structural floor that consumes exclusively CPU; no GPU or NPU can offload protocol parsing, asymmetric signature verification, token introspection, or registry lookup at any practical scale. The optimizable remainder ρ_{rt} is the component over which runtime engineering has direct leverage. Empirical anchoring for the magnitude of $\rho_{net} + \rho_{sec}$ comes from the MeshInsight study of service mesh sidecar overhead [9], which measured CPU usage increases of 42 to 163 percent from service mesh deployment in production microservice applications, with protocol parsing accounting for 63 to 77 percent of total sidecar CPU overhead.

A.2 Composition of the Optimizable Remainder

The optimizable remainder ρ_{rt} itself bundles several per-call CPU consumers that are structurally CPU-bound but admit varying degrees of runtime optimization. Inter-agent observability (distributed tracing context propagation, structured logging, metrics emission) costs roughly 5 to 30 μs per call at modest sampling rates; the in-process portion (span lifecycle, attribute serialization, exporter batching) is irreducible while sampling rate sets the optimizable share. Distributed state management (conversation context serialization, session persistence, cross-replica synchronization for KV-store-backed agent state) costs 20 to 100 μs per call for typical retrieve-update-persist sequences. Routing and orchestration logic (model tier selection, escalation decisions, circuit breaker state evaluation, retry budgeting, rate limit checks) adds 5 to 30 μs per call, applied to every routing decision; agentic systems make more such decisions per user-task than monolithic systems because each agent involves a routing step. Pre- and post-processing (tokenization, output detokenization, schema validation for tool-calling responses, prompt template assembly, context window construction with truncation) costs 50 to 200 μs

per call for typical 2 KB contexts. Policy enforcement (PII detection, capability checks, output safety filtering, content guardrails) costs 20 to 200 μs per call depending on whether the check is rule-based or invokes a small classifier model. Each of these has a structural minimum that runtime engineering cannot eliminate, reinforcing the CPU-weighted compute distribution rather than offsetting it.

A.3 Background CPU Load

In addition to per-call costs, agentic systems incur a background CPU load that is independent of the per-call rate but present continuously and largely absent from monolithic single-process deployments. Agent lifecycle (cold-start of new agent instances, warm-up curves as caches populate, scaling decisions, graceful shutdown) consumes 100 to 500 ms of mostly CPU work per cold start, which becomes a continuous baseline when agents come and go on minute-scale timescales as serverless lifecycle behavior suggests. Runtime housekeeping (garbage collection in managed runtimes such as Node.js, Python, or the JVM; event loop scheduling; thread pool management; memory allocator overhead) adds a baseline CPU draw that scales with the per-agent process boundary count. Thermal and power management on end devices runs on CPU and reacts to telemetry that also runs on CPU. In multi-device deployments, cluster consensus protocols (leader election, configuration consistency, gossip about node health, capacity advertising, rebalancing decisions) operate continuously across all cluster members. None of these consumers exist in a monolithic single-process stack at the same intensity, and all of them reinforce the CPU-weighted compute distribution of agentic deployments.

A.4 NPU Usage in Agentic Workloads

The shift toward NPU usage, smaller in absolute terms than the CPU shift but structurally significant, arises from a class of workloads that exists in agentic systems but is largely absent from monolithic LLM serving. Always-on sensing components (voice activity detection, wake-word detection, motion and gesture triggers, environmental change detection) are typically TinyML models that fit comfortably on the NPU and run continuously rather than on demand. Small classifier models embedded in the routing path (intent classification, language detection, toxicity detection, sentiment, topic routing) provide per-call routing decisions that monolithic LLM serving does not require; each such classifier is NPU-eligible. Per-call vector embedding for retrieval-augmented generation, when the embedding model fits in the 100M to 500M parameter range, runs on the NPU with low overhead and is invoked on every reasoning call rather than only during bulk indexing. Output safety filters and content guardrails frequently use small classifier models on the NPU. Sensor fusion and event detection over multiple sensor streams also map cleanly to NPU operators where the operator set permits. The aggregate effect is that agentic systems exercise NPU resources more uniformly than monolithic systems, where the NPU is typically idle except during dedicated quantized inference phases.

A.5 Per-Call Operational Cost Model

For each inter-agent call carrying payload of p bytes, the per-endpoint CPU work is:

$$c_{call}(p) = (\alpha_{ser} + \alpha_{crypto}) \cdot p + (\beta_{net} + \beta_{auth} + \beta_{sign} + \beta_{disc}) \tag{10}$$

The per-byte term combines protocol parsing (α_{ser} at 0.5 to 2 ns per byte for HTTP or gRPC) and symmetric encryption with hardware acceleration (α_{crypto} at 0.5 to 2 ns per byte for AES-256-GCM with AES-NI or ARMv8 Crypto Extensions). The per-call fixed term combines network stack traversal (β_{net} at 10 to 50 μs), mutual TLS amortized over connection reuse (β_{auth} at 50 to 150 μs equivalent per call), token validation (10 to 200 μs depending on cache state), message signature verification (β_{sign} at 40 to 200 μs for Ed25519 or ECDSA P-256),

and the structural floor of discovery cost (β_{disc} at 30 to 180 μs depending on cache hit rate and registry topology). Cross-device coordination at cluster scale adds further depth to β_{net} and β_{disc} as both grow with the number of cluster members and the topology of the agent registry. The total operational CPU demand combines the per-call cost with the call rate: $W_{coord} = 2 \cdot N \cdot \lambda \cdot c_{call}(\bar{p})$ where λ is the per-agent inter-agent call rate and $\bar{p}$ is the mean payload, and the factor of two reflects that both sender and receiver pay the per-call cost.

References

[1] X. Wei, J. Zhang, H. Li, J. Chen, R. Qu, M. Li, X. Chen, and G. Luo, *Agent.xpu: Efficient Scheduling of Agentic LLM Workloads on Heterogeneous SoC*, arXiv:2506.24045, 2025.

[2] H. Zhang and J. Huang, *Challenging GPU Dominance: When CPUs Outperform for On-Device LLM Inference*, arXiv:2505.06461, 2025.

[3] J. Xiao, Q. Huang, X. Chen, and C. Tian, *Understanding Large Language Models in Your Pockets: Performance Study on COTS Mobile Devices*, arXiv:2410.03613, 2024.

[4] D. Xu, H. Zhang, L. Yang, R. Liu, G. Huang, M. Xu, and X. Liu, *Fast On-device LLM Inference with NPUs*, in Proc. ACM ASPLOS '25, 2025. arXiv:2407.05858.

[5] S. M. Alamouti, F. Arjomandi, M. Burger, and J. Hsu, *App-Native vs. Agentix-Native Architectures for On-Device AI Agents: A Quantitative Engineering Guide Using Device-First Continuum AI*, mimik Technology Inc., 2025.

[6] S. M. Alamouti, F. Arjomandi, M. Burger, and H. Gün, *Device First Continuum AI (DFC-AI): Realizing human-like AI*, in Proc. Future Technologies Conf. (FTC) 2025, Lecture Notes in Networks and Systems, vol. 1676, Springer, 2026, doi:10.1007/978-3-032-07989-3_31.

[7] AMD, *Ryzen AI Embedded P100 and X100 Series Product Documentation*, 2026. https://www.amd.com/en/newsroom/press-releases/2026-1-5-amd-introduces-ryzen-ai-embedded-processor-portfol.html

[8] NVIDIA, *Jetson AGX Thor Series (T5000) Technical Brief and Product Documentation*, 2026. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-thor/

[9] X. Zhu, G. She, B. Xue, Y. Zhang, Y. Zhang, X. K. Zou, X. Duan, P. He, A. Krishnamurthy, M. Lentz, D. Zhuo, and R. Mahajan, *Dissecting Overheads of Service Mesh Sidecars*, in Proc. ACM Symposium on Cloud Computing (SoCC), 2023.