Beyond the Computer: The Noeton

A Model-Native Architecture for the AI Era

JohnRothan*

June 2026

Abstract

For seventy years the computer has been a von Neumann machine: a central processor that fetches instructions and data, word by word, across a bus from a passive store, surrounded by peripherals. That contract has survived every device generation from the mainframe to the smartphone. The arrival of large language models (LLMs) as a general substrate for computation strains it to the breaking point. Foundation-model inference is dominated not by arithmetic but by the movement of enormous, mostly-static weight tensors and a growing key–value cache; on commodity hardware the autoregressive decode phase is bound by memory bandwidth, not by floating-point throughput. The industry's present answer—the "AI PC," a von Neumann machine with a neural processing unit (NPU) bolted on—treats the model as one more accelerated subroutine. We argue this is a transitional form, not the terminal one.

We propose to redesign the machine, from the hardware up, around the model. We call the result a Noeton (from *noesis*, the act of understanding), and its organizing principle the *Model-Native Architecture* (Mna), posed as the successor to the von Neumann machine. The Noeton rests on five principles: (i) **memory-centric** organization, in which model weights and the key–value cache are first-class citizens served by processing-in-memory and disaggregated, pooled memory rather than streamed through a core; (ii) **model-native compute**, a tensor/attention dataflow fabric in which a conventional CPU is demoted to a housekeeping orchestrator; (iii) **model-as-operating-system**, where an LLM is the kernel, agents are processes, tools are system calls, and a vector store is the file system; (iv) **network-mandatory, tiered cognition**, where an on-device small model provides only basic, degraded-mode function and full capability is drawn from networked weight pools—so that, deliberately, an offline Noeton can do little; and (v) **intent-driven interaction**, in which the user's unit of work is an expressed intent over context rather than an instruction over data. We ground each principle in existing hardware and systems research—the memory wall, processing-in-memory, CXL memory pooling, wafer-scale integration, deterministic dataflow processors, neuromorphic chips, and the emerging "LLM OS"—and we synthesize them into a single coherent reference design with a roofline- and energy-based motivation. We do not build a Noeton; this is a position paper. We offer it as a target worth arguing about, and we take seriously its costs: mandatory connectivity trades autonomy, privacy, and equity for capability, and we devote a full section to those risks rather than waving them away.

Keywords: computer architecture · large language models · memory wall · processing-in-memory · memory disaggregation · domain-specific architecture · LLM operating system · edge–cloud inference.

*Draft prepared for arXiv (primary: cs.AR; cross-list: cs.AI). Comments welcome.

1 Introduction

The computer we use today is, in its bones, the machine John von Neumann and his collaborators described in 1945: a central processing unit that fetches a stream of instructions and operands, one word at a time, from a single passive memory, ringed by input/output peripherals. The abstraction has been extraordinarily durable. It survived the move from vacuum tubes to transistors to integrated circuits, from the mainframe to the minicomputer to the personal computer to the smartphone. Caches, pipelines, out-of-order execution, multicore, and GPUs are all elaborations *within* the contract, not departures from it. We still program a stored-program computer; we still think of *compute* as the scarce resource and memory as the place we keep things between computations.

Two things have changed at once. First, the historical engine of the contract's success—the steady, "free" improvement in single-thread performance from Dennard scaling—has stopped. The end of Dennard scaling and the onset of *dark silicon* [1] mean that we can no longer power all the transistors we can fabricate, and the field's response has been a turn toward *domain-specific architectures*: silicon co-designed with a particular workload rather than general-purpose silicon asked to do everything [2]. Second, and more consequentially, a single workload has become important enough to design a machine around. The Transformer [3] and the large language models built on it [4, 5] have become a *general substrate*: a single learned function that drafts code, answers questions, drives agents, controls tools, and increasingly mediates how people interact with machines at all. For a growing fraction of what computers are asked to do, the model *is* the application.

Put those two facts together and an obvious question follows: if we are going to build domain-specific machines, and if the dominant domain is now the foundation model, what does a machine designed *natively* for foundation models look like? The industry has a provisional answer already on shelves—the "AI PC" [6, 7]: an ordinary von Neumann laptop with a neural processing unit (NPU) added beside the CPU and GPU. We will argue (Section 3) that this is a transitional form. It accelerates the model, but it still treats the model as a guest—one more subroutine dispatched by a CPU through a memory hierarchy that was designed for a different era's working set. The deeper mismatch is left untouched.

The mismatch. The defining cost of foundation-model inference is not arithmetic; it is data movement. A modern LLM is tens to hundreds of gigabytes of weights that must be read for every token generated, plus a key–value (KV) cache that grows with the conversation and must be re-read on every step. The result is that the generation (decode) phase of inference is *memory-bandwidth-bound*: the arithmetic units sit idle waiting for operands [8, 9]. This is the modern face of an old problem, the "memory wall" [10], and it has only widened: peak compute has grown far faster than memory bandwidth, so for LLMs the bottleneck is now firmly on the memory side of the ledger [11]. A roofline analysis [12, 13] places autoregressive decoding far below the machine's compute ceiling, pinned against its bandwidth ceiling. The von Neumann organization—compute in the center, data shuttled in from the edges—is precisely the wrong shape for this workload.

This paper. We take the question seriously and answer it with a design, not a benchmark. We propose a machine organized around the model from the hardware up. Because so little of what defines it is recognizably a "computer" in the stored-program sense—the central scarce resource is memory, not compute; the kernel is a learned model, not a hand-written executive; its native unit of work is an intent, not an instruction—we give it a different name. We call it a Noeton, from the Greek *noesis*, the act of understanding or intellection, and we call its organizing discipline the *Model-Native Architecture* (Mna). We mean the name to stand in the same relation to the coming

era that "von Neumann machine" stands to the one now ending:

$$\text{von Neumann machine} \longrightarrow \textsc{Noeton}.$$

Contributions.

1. **A diagnosis** (Sections 2 and 3): we argue, via the memory wall and a roofline of LLM inference, that the bolt-on "AI PC" inherits exactly the bottleneck it should remove, and is therefore not the terminal form of the AI-era machine.

2. **Five design principles** (Section 4) for a model-native machine: memory-centric organization, model-native compute, model-as-OS, network-mandatory tiered cognition, and intent-driven interaction.

3. **A reference architecture** (Section 5): the Noeton, a five-layer stack—memory-centric substrate, tensor-native compute fabric, a Model-OS runtime, a mandatory network fabric with edge–cloud tiered cognition, and an intent interface—each layer instantiated with existing hardware and systems research rather than speculation.

4. **A quantitative motivation** (Section 6): a roofline and data-movement-energy argument for why the memory-centric inversion wins, with stated assumptions.

5. **An honest accounting** (Section 8) of what the design costs: mandatory connectivity converts a personal machine into a networked dependency, with consequences for privacy, autonomy, equity, security, and energy that we treat as first-order, not footnotes; and a discussion (Section 9) of why the artifact deserves a new name.

We want to be clear about epistemic status. This is a position and vision paper in the genre of [2] and the LLM-OS proposals [14]; it builds no hardware and reports no measurements of a Noeton, because none exists. Its claim is that the pieces already exist in the literature, that they point in a consistent direction, and that naming and assembling that direction is itself useful work.

2 Background and Motivation

Our argument rests on three observations from the existing literature: that the von Neumann organization has a structural bottleneck that has been known for decades; that this bottleneck is, for LLM inference specifically, the dominant cost rather than a secondary one; and that the field has independently developed most of the hardware ingredients needed to escape it, without yet assembling them into a coherent machine. We review each in turn.

2.1 The von Neumann bottleneck and the memory wall

The structural critique is as old as the architecture's maturity. Backus, in his 1977 Turing lecture, named the "von Neumann bottleneck": the single channel between processor and store down which the entire computation must be pumped, word by word, which he argued was not only a performance limit but an *intellectual* one, shaping how we are able to think about programs [15]. The performance face of the same problem was quantified by Wulf and McKee in 1995 as the "memory wall" [10]: because processor speed was improving far faster than memory latency, average memory-access time would eventually dominate execution time regardless of how fast the processor became. Their arithmetic was simple and their conclusion uncomfortable—the wall was not a question of *if* but *when*.

The intervening decades blunted the wall with a deep cache hierarchy, which works precisely to the extent that a program reuses data that fits in fast memory. The energy cost of the underlying

movement, however, never went away: moving a word from DRAM costs on the order of two to three magnitudes more energy than the arithmetic operation that consumes it [16]. For workloads with high *arithmetic intensity*—many operations per byte fetched—this is tolerable, because the fetch is amortized. The trouble, as we now show, is that LLM inference has the opposite character.

2.2 LLM inference is memory-bound

A foundation model in deployment spends its time in two regimes. During *prefill*, it ingests the prompt and can process all tokens in parallel; this phase has high arithmetic intensity and is compute-bound. During *decode*, it generates the response one token at a time, and each new token requires reading the *entire* weight matrix once and the *entire* growing KV cache once, to produce a single column of activations [9]. The arithmetic intensity of decode is therefore very low—roughly one multiply-accumulate per weight byte read at batch size one—and the phase is bound by memory bandwidth, not by the arithmetic units, which stall waiting for operands [8]. This is not a tuning detail; it is the central economic fact of serving LLMs, and it is why so much systems work targets the KV cache specifically (e.g., multi-query and grouped-query attention shrink the cache to relieve exactly this bandwidth pressure [8]).

The roofline model [12] makes the situation visual: plotting attainable performance against arithmetic intensity, a machine has a sloped bandwidth ceiling and a flat compute ceiling that meet at the "ridge point." Workloads to the left of the ridge are bandwidth-bound; those to the right are compute-bound. Recent work places LLM decoding firmly and far to the left of the ridge on commodity accelerators [13]: the expensive floating-point silicon we paid for is mostly idle, gated by how fast weights and cache can be delivered. We reproduce this picture schematically in Figure 4 (Section 6). The crucial point for this paper is *architectural*: when the binding constraint is the delivery of data to the compute units, an organization that places compute at the center and data at the periphery is optimizing the wrong thing.

2.3 The bottleneck has only widened

One might hope the wall is receding. It is not. Gholami et al. [11] track the relevant growth rates and find that peak hardware compute (FLOP/s) has grown far faster than either DRAM bandwidth or interconnect bandwidth across recent accelerator generations. The gap between what the arithmetic units can consume and what memory can supply is therefore *widening* with each generation, not closing. As models continue to scale and context windows lengthen—both well-motivated by the empirical scaling laws and emergent-capability literature [5, 17] (whose strong reading is contested [18], though the systems pressure is real regardless of interpretation)—the working set that must be moved per token grows, and the mismatch deepens. A machine designed for the next decade of this workload cannot inherit the von Neumann data-movement assumption and expect to scale.

2.4 The ingredients of an escape already exist

The encouraging half of the picture is that computer architecture has, over the last decade, produced most of the pieces required to build a memory-centric, model-native machine—but as separate research programs, not as an integrated whole. *Processing-in-memory* (PIM) moves computation to where the data lives, attacking the data-movement cost at its root [19, 20], and commercial HBM-PIM [21] demonstrates that this is manufacturable silicon, not simulation. *Memory disaggregation* and CXL-based memory pooling [22, 23] let memory capacity be allocated independently of compute, which is exactly what a machine whose scarce resource is weight capacity wants. *Wafer-scale integration* [24] keeps an entire model resident in on-die SRAM, sidestepping off-chip bandwidth altogether. *Deterministic dataflow processors* [25, 26] and the original tensor accelerators [27] show what compute looks like when it is shaped to the tensor workload rather than to general-purpose

code. And the systems community has begun to treat the model itself as an operating system [28–30]. Each of these is a fragment of the same machine. The contribution of this paper is to argue that they are fragments of *one* machine, and to draw it.

3 Why the "AI PC" Is Not the Terminal Form

It would be unfair to design a strawman. The current generation of AI-capable personal machines is a genuine and useful engineering advance, and any honest proposal must explain precisely where it falls short rather than dismiss it. This section does that.

3.1 What the AI PC actually is

The "AI PC," as shipped, is a von Neumann computer with a third compute engine added beside the CPU and GPU: a neural processing unit (NPU) specialized for the matrix and convolution kernels of on-device inference, marketed with a trail-of-operations-per-second figure of merit [6]. Apple's variant pairs an on-device model with a larger server model, escalating to the cloud when the local model is insufficient [7, 31]. Both are real improvements: they bring useful inference to the device, cut latency for small models, and keep some data local. We are not claiming they do nothing. We are claiming they leave the architecture's center of gravity exactly where it was.

3.2 Three inherited mismatches

1. The CPU still owns the machine. In the AI PC the NPU is a *device*: the CPU runs the operating system, schedules work, owns the memory map, and dispatches inference jobs to the NPU as it would to any accelerator. The model is a guest invoked by a host. But for a machine whose primary job is to run a model, this inverts the natural hierarchy. The model is not a subroutine the operating system occasionally calls; it is, or should be, the thing the operating system is *for*. Keeping a general-purpose sequential core as the master and the model as a callee preserves a control structure built for a world in which the dominant workload was branchy, sequential application code—a world that, for an increasing share of usage, no longer describes what the machine is doing.

2. The memory hierarchy is still compute-centric. The NPU draws its operands through the same memory hierarchy as everything else: weights stream from DRAM, across a bus, into the accelerator, for every token. The bolt-on adds arithmetic throughput—more FLOP/s—to a workload whose binding constraint we established in Section 2.2 is *not* FLOP/s but bandwidth. Adding compute to a bandwidth-bound workload raises the compute ceiling the workload never reaches while leaving the bandwidth ceiling that actually binds it untouched. The AI PC therefore inherits the memory wall in full; it has merely made the idle arithmetic units faster. This is the single most important reason we regard it as transitional: it accelerates the part that was not the bottleneck.

3. Connectivity is an afterthought, not a contract. The AI PC treats the network as an optional escalation path: do what you can locally, reach for the cloud when you must [7]. This is a sensible engineering compromise, but it is a compromise within the old contract, in which a personal computer is a self-sufficient machine that *may* use the network. We will argue (Section 4, Section 5.4) that an AI-era machine is better understood the other way around: as a thin, local manifestation of a fundamentally networked cognitive resource, for which offline operation is a deliberately degraded mode rather than the default. The AI PC does not make that choice; it keeps the local machine primary and bolts cognition, like everything else, onto the side.

3.3 The pattern: acceleration within the contract

Each mismatch is an instance of the same thing. The AI PC accelerates the model *within* the von Neumann contract instead of rebuilding the machine around the model. This is the historically normal first response to a new dominant workload—graphics got a bolt-on GPU before GPUs reshaped whole datacenters; we should expect inference to follow the same arc—but it is a first response, not a final one. The thesis of this paper is that the final response inverts the relationship: the model stops being a device the computer accelerates and becomes the organizing center the machine is built to serve. The rest of the paper describes that machine.

4 Design Principles of a Model-Native Machine

Before describing the NOETON concretely, we state the five principles that generate it. Each is a deliberate inversion of a von Neumann assumption, and each is independently supported by existing work; the architecture in Section 5 is what one gets by taking all five seriously at once. We state them as a design manifesto, then build the machine to satisfy them.

P1 — Memory-centric, not compute-centric. *The scarce, organizing resource is memory, and the architecture should make weights and the KV cache first-class citizens.* The von Neumann machine treats memory as passive storage from which a central processor draws operands. The NOETON treats the model's parameters and working cache as the primary inhabitants of the machine, served *in place* by computation brought to the data—processing-in-memory [19, 21]—and scaled by pooled, disaggregated memory [22] rather than by streaming through a core. Compute density follows memory, not the reverse. The justification is Section 2.2: when the workload is bandwidth-bound, minimizing data movement is the whole game.

P2 — Model-native compute; the CPU is demoted. *The dominant compute fabric is a tensor/attention dataflow engine; a conventional sequential core survives only as a housekeeping orchestrator.* Where the AI PC keeps a general-purpose CPU as master and the model as guest (Section 3), the NOETON inverts the roles. The tensor and attention operators are the native instructions of the machine, executed on a deterministic dataflow fabric [25–27]; a small CPU handles boot, I/O glue, and control flow that does not belong in the model. The sequential core does not disappear—something must mind the housekeeping—but it stops being the center.

P3 — The model is the operating system. *An LLM is the kernel; agents are processes, tools are system calls, and a vector store is the file system.* The user-facing runtime of the NOETON is not a hand-written executive that occasionally calls a model; it is the model, acting as the privileged interpreter that schedules work, manages context, invokes tools, and mediates I/O. This reframing is already underway in systems research [14, 28–30]; we adopt it as a hardware-design constraint, because a machine whose kernel is a model should provide the model the privileges, memory abstractions, and protection domains a kernel needs.

P4 — Network-mandatory; cognition is tiered. *Full capability lives in the network; the device hosts only a small model that provides basic, degraded-mode function when offline.* This is the most contentious principle and we state it deliberately. The NOETON is not a self-sufficient computer that may use the network; it is the local endpoint of a networked cognitive resource. An on-device small model supplies a floor of capability—basic dictation, simple queries, local control—so the machine is not a brick when disconnected, but the large models and the pooled weight memory that constitute full capability are reached over the network, with inference split across device,

edge, and cloud [7, 32]. This is a return, at a new layer, to the old thesis that "the network is the computer" [33]. We treat the autonomy, privacy, and equity costs of this choice as first-order and confront them in Section 8; here we only assert that the design *makes* the choice rather than leaving connectivity an afterthought.

P5 — Intent-driven, not instruction-driven. *The unit of work is an expressed intent over context, not an instruction over data.* The von Neumann programming model presents the machine with a precise sequence of operations on explicitly addressed data. The NOETON's native interface is an intent—a goal stated in natural language or other modality—resolved against an ambient context the machine maintains. The "program" is increasingly the weights; the "bus traffic" is increasingly tokens; the user's job is to specify *what*, and the model's job is to determine *how*. This principle shapes the interface and the instruction set abstraction (Section 5.5) rather than the silicon directly, but it is what makes the other four cohere into a machine a person actually uses.

These five are not independent wishes; they reinforce one another. Memory-centric organization (P1) is what makes a model-as-kernel (P3) affordable, because the kernel's state—its weights and context—is exactly what the memory system is built to hold. Tiered cognition (P4) is what lets a single small device present the interface of a very large model, which is what an intent-driven interaction model (P5) implicitly promises. And demoting the CPU (P2) is the structural change that stops the old contract from quietly reasserting itself. The next section assembles them.

5 The Noeton Architecture

We now assemble the five principles into a concrete reference design. The NOETON is organized as five layers, shown in Figure 1, and contrasted with the von Neumann organization in Figure 2. From the bottom: a *memory-centric substrate* holds and serves the model; a *tensor-native compute fabric* executes its operators; a *Model-OS runtime* acts as the kernel; a *network fabric* extends the machine into tiered edge–cloud cognition; and an *intent interface* presents it to the user. We describe each layer, naming the existing technology that instantiates it so that the design is a synthesis of demonstrated parts, not a wish list.

5.1 Layer 1: The memory-centric substrate

The foundation of the machine is a memory system designed to *hold a model and compute on it in place*, realizing principle P1. It combines three existing techniques that today live in separate products.

Processing-in-memory (PIM) embeds arithmetic at the memory banks so that the bandwidth-bound parts of inference execute without moving weights off the memory die. General-purpose PIM is real, shipping silicon—UPMEM's DRAM-resident processors [20] and Samsung's HBM-PIM, with MAC/SIMD units at the HBM bank level reaching internal bandwidths far above the external pins [21]—and recent work shows the specific win for LLMs: AttAcc offloads the bandwidth-bound attention layer to PIM while leaving the compute-bound feed-forward layers on a conventional accelerator, reporting large performance and energy improvements on a 175B-parameter model [34]. The NOETON generalizes this: the bandwidth-bound operators (attention over the KV cache, and weight-streaming matrix–vector products in decode) are PIM-native, executed where the data lives. For ultra-low-energy inference, the substrate may include analog in-memory crossbars, which have been shown to recover full-precision accuracy on Transformer-class networks despite device non-idealities through hardware-aware training [35, 36].

Disaggregated, pooled weight memory provides capacity. A machine whose scarce resource is parameter capacity wants to scale memory independently of compute, which is exactly what

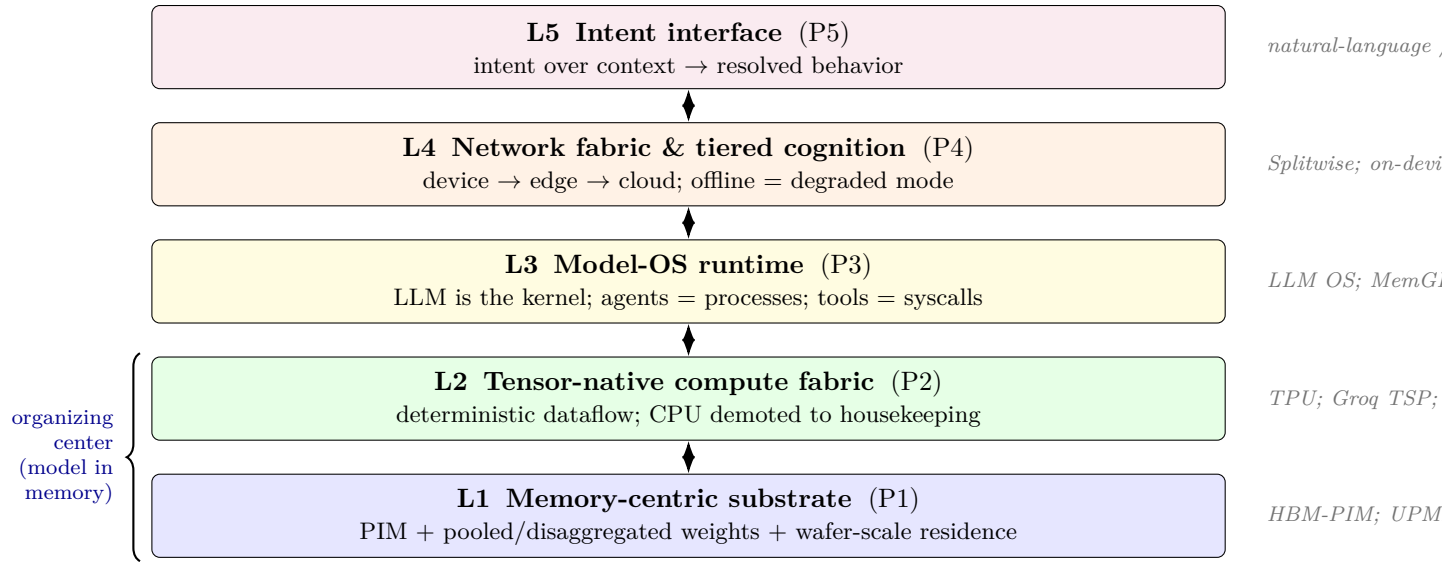

Figure 1: The NOETON five-layer stack. Unlike the von Neumann machine, the organizing center is the memory-resident model (bottom two layers), not a central processor. Each layer is annotated with the existing technology that instantiates it.

CXL-based memory pooling delivers [22, 23], with operating-system support for placing pages across the tiers [37]. In the NOETON, model weights live in a pooled tier that the device draws from on demand: a small model resides locally, larger models are paged in from an edge or cloud weight pool. This is the hardware substrate of the tiered cognition of P4.

Wafer-scale on-die residence is the high-end extreme of the same idea: keep the entire model in on-die SRAM so that off-chip bandwidth never binds at all. The Cerebras wafer-scale engine demonstrates this for training and inference, with weight-streaming scale-out that decouples weight storage from the compute substrate [24, 38, 39]. The NOETON treats wafer-scale residence and CXL pooling as two ends of a capacity spectrum: the same architecture, parameterized by how much of the model is resident versus paged.

5.2 Layer 2: The tensor-native compute fabric

Above the substrate sits the compute fabric that executes the model's operators, realizing principle P2. Its native "instructions" are tensor and attention primitives, not scalar arithmetic. The lineage is well established: the systolic matrix engine of the TPU showed an order-of-magnitude performance-per-watt advantage by shaping silicon to dense matrix multiply [27]; attention-specific accelerators such as A^3 [40] and SpAtten [41] showed further gains by exploiting the structure and sparsity of attention directly in hardware; and deterministic dataflow processors—Groq's tensor streaming processor [25] and SambaNova's reconfigurable dataflow unit [26]—showed that removing reactive caches and arbiters in favor of compiler-scheduled dataflow yields predictable, low-latency inference, which matters for an interactive cognitive machine.

The NOETON's fabric is a deterministic, statically-scheduled dataflow array, tightly coupled to the PIM substrate beneath it: operators that are compute-bound (prefill matrix multiplies, feed-forward layers) run on the fabric at high arithmetic intensity, while operators that are bandwidth-bound (decode-time attention and weight-streaming) are pushed down into the substrate (P1). The split is the AttAcc insight [34] elevated to an architectural principle: *place each operator where its binding*

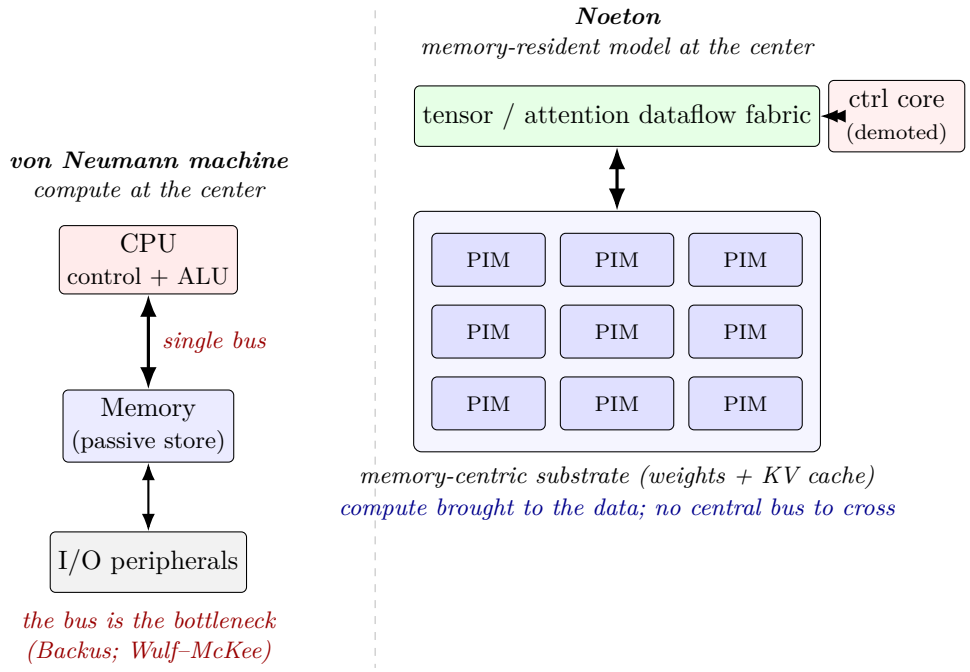

Figure 2: Von Neumann machine (left) versus NOETON (right). The von Neumann machine routes all data through a central processor over a single bus—the bottleneck named by Backus [15] and Wulf–McKee [10]. The NOETON inverts the figure: compute is distributed into a memory-resident model substrate, and a demoted control core sits at the edge.

resource is cheapest. A conventional sequential core remains on the die, but in the demoted role of P2—it boots the machine, drives peripherals, and runs the thin control logic that is genuinely branchy and sequential, while the model's actual work happens on the fabric–substrate pair.

5.3 Layer 3: The Model-OS runtime

The kernel of the NOETON is a model, realizing principle P3. Rather than a hand-written executive that schedules processes and calls a model as a service, the privileged runtime *is* an LLM that interprets intent, plans, invokes tools, and manages its own context. This is the "LLM OS" framing made concrete [28]: the model is the kernel process; the context window is managed like virtual memory, with relevant material paged in and out of a fast working context from slower external stores, exactly as MemGPT demonstrates [29]; agents are the processes the kernel schedules; and tool invocations are the system calls by which the model reaches the outside world [14, 30]. Table 2 (Section 7) gives the full correspondence between classical OS concepts and their Model-OS counterparts.

Making the model the kernel imposes hardware requirements that the lower layers are built to meet. The kernel's state—its weights and active context—is large and must be served at high bandwidth, which is why Layer 1 is memory-centric. The kernel must be able to extend its effective memory beyond what is resident, which is why Layer 1 is disaggregated and paged. And the kernel must enforce protection between agent-processes and mediate their access to tools, which means the NOETON needs hardware protection domains defined over *model context and tool capabilities*, not only over memory pages—a research problem we flag in Section 8 rather than claim to solve.

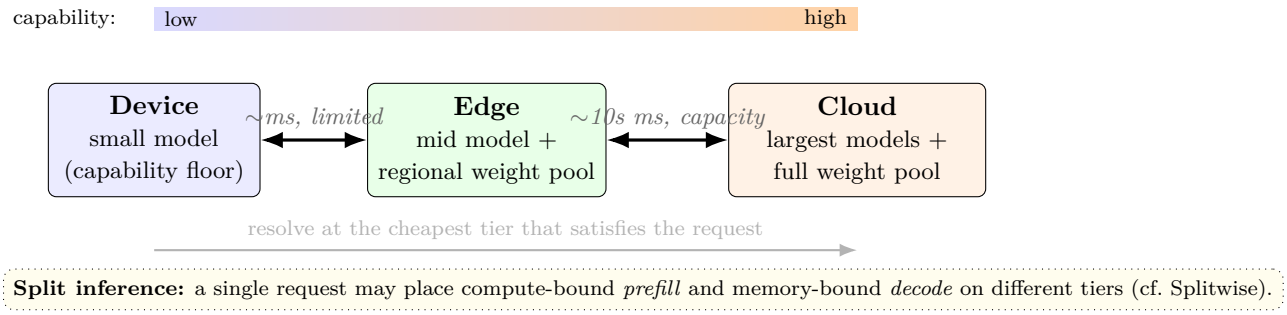

Figure 3: Tiered cognition (P4). A request is resolved at the cheapest tier that can satisfy it; a single inference may be split across tiers (e.g., prefill vs. decode). The on-device small model defines the offline *degraded mode*: the machine is usable but deliberately limited when disconnected.

5.4 Layer 4: The network fabric and tiered cognition

The fourth layer makes the machine networked by construction, realizing principle P4 and shown in Figure 3. Cognition is tiered across three scales. On the device, a small model provides a guaranteed floor of capability and the lowest-latency path. At the edge, a mid-sized model and a regional weight pool serve the common case at low latency. In the cloud, the largest models and the full weight pool provide maximum capability. A request is resolved at the cheapest tier that can satisfy it, and a single inference may be *split* across tiers—for example, separating the compute-bound prefill from the memory-bound decode and placing them on differently-provisioned hardware, as Splitwise does within a datacenter [32] and as device–edge–cloud split-inference schemes do across the network [7, 31].

The defining commitment of this layer is the *degraded mode*. When the NOETON is offline, it falls back to the on-device small model: basic dictation, local search, simple control and drafting remain available, but the full cognitive capability that defines the machine does not. This is a deliberate inversion of the personal-computer contract. We are aware that "your computer barely works without a network" reads as a defect; Section 8 argues it is an honest exposure of a dependency that the AI PC's optional-escalation model merely hides, and confronts its real costs. The intellectual ancestor of this layer is Sun's 1980s thesis that "the network is the computer" [33], and its nearest shipping relative is the cloud-first thin client; the NOETON differs in that what lives in the network is not files or applications but *cognition* itself.

5.5 Layer 5: The intent interface and instruction abstraction

The top layer presents the machine to the user and defines its instruction-set abstraction, realizing principle P5. The classical interface offers the machine an instruction stream over addressed data; the NOETON offers it an *intent* over ambient context. The user specifies a goal—in language, speech, or another modality—and the Model-OS kernel resolves it against the context it maintains, decomposing it into operators on the fabric, tool calls, and, where needed, escalations across the network tiers. In this abstraction the analogues of the classical machine shift: the "program" is principally the model weights; the "instruction stream" is a sequence of tokens; "addressing" is retrieval over a vector store rather than indexing into linear memory. Agents installed on the machine are its applications, scheduled by the kernel as processes (Table 2). The contribution of

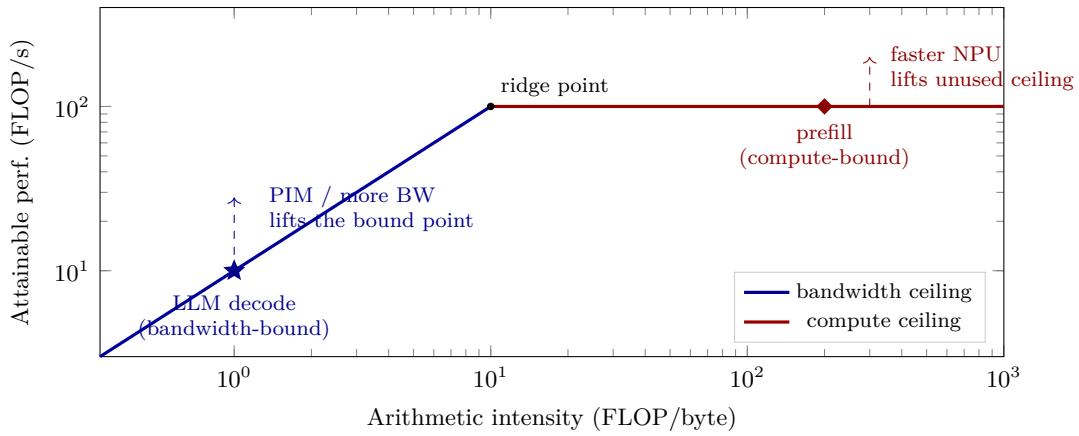

Figure 4: Roofline schematic. Prefill (high arithmetic intensity) is compute-bound, to the right of the ridge point. Decode at small batch is pinned one to two orders of magnitude to the *left*, bandwidth-bound: the compute ceiling the machine paid for is unreachable. Raising the compute ceiling (a faster NPU) does not move the decode point up; raising the bandwidth ceiling, or reducing bytes moved per token via PIM, does.

this layer is not a new widget toolkit but a coherent answer to "what is the machine's instruction set, now that the instruction set is a model?"—namely, intent in, resolved behavior out, with the lower four layers existing to make that resolution fast, large, and networked.

6 Quantitative Motivation

The case for the NOETON does not rest on prose alone. This section gives a first-order, back-of-the-envelope argument—explicitly approximate, with stated assumptions—for why the memory-centric inversion is the right move for the LLM workload. We make no claim to a cycle-accurate evaluation of a machine that does not exist; we claim only that the order-of-magnitude reasoning points unambiguously in one direction.

6.1 A roofline of LLM inference

Consider decoding a single token from a dense Transformer with P parameters stored at b bytes each, at batch size B. To produce one token, the machine must read essentially all P parameters once, transferring $P{\cdot}b$ bytes, and performs roughly $2P$ floating-point operations per sequence in the batch, i.e. $2PB$ in total. The arithmetic intensity—operations per byte of weight traffic—is therefore

$$I_{\text{decode}} \approx \frac{2PB}{Pb} = \frac{2B}{b}.$$

The striking feature is that I_{decode} is independent of model size P: making the model bigger moves more bytes and does proportionally more arithmetic, leaving intensity fixed. For interactive serving at small batch (B of order one) and 16-bit weights ($b = 2$), I_{decode} is of order one operation per byte. A contemporary accelerator has a ridge-point intensity (peak FLOP/s divided by peak bandwidth) of order 10^2 operations per byte. Decode therefore sits one to two orders of magnitude to the *left* of the ridge: deep in the bandwidth-bound region, with attainable performance set entirely by bandwidth and the expensive arithmetic units largely idle. Figure 4 shows this schematically; the quantitative version is developed in detail by [9, 13].

Two consequences follow directly. First, *adding FLOP/s does not help decode.* Raising the compute ceiling—the AI PC's bolt-on NPU—lifts a ceiling the workload never touches. Only raising

the effective bandwidth, or reducing the bytes that must be moved per token, lifts the decode point. Second, the two levers that do help are exactly the NOETON's substrate (P1): processing-in-memory reduces the bytes that cross any external boundary by computing where the weights live [21, 34], and on-die or in-package residence (wafer-scale, HBM) raises the bandwidth that binds the point [24]. The architecture is, in this precise sense, derived from the roofline rather than asserted.

6.2 The energy ledger

The bandwidth argument has an energy twin. The dominant energy cost in this workload is not the arithmetic but the movement of the operands to it: a DRAM access costs on the order of 10^2 to 10^3 times the energy of the multiply-accumulate it feeds [16]. Because decode reads the entire parameter set per token, per-token energy is dominated by weight *transport*, and a machine that transports weights across a board for every token of every user is paying that premium continuously. Moving the computation to the data—PIM and on-die residence—attacks the dominant term directly, which is why memory-centric designs report energy improvements rather than merely performance improvements [34, 42]. At fleet scale this is not a microarchitectural nicety: inference, not training, dominates the lifecycle energy of a deployed model, and LLM serving already represents a material and growing share of datacenter demand [43–45]. An architecture that reduces bytes-moved-per-token reduces joules-per-token, and the two are, to first order, the same quantity.

6.3 What the analysis does and does not show

We are deliberate about the limits of this argument. It shows that the *binding constraint* of the dominant workload is bandwidth and data-movement energy, and that the NOETON's memory-centric substrate targets exactly that constraint while the bolt-on accelerator targets a non-binding one. It does *not* show a specific speedup or efficiency number for a NOETON, because that depends on parameters of a machine no one has built—the PIM-to-fabric bandwidth, the resident-versus-paged fraction, the network tier latencies. Those are precisely the quantities a future evaluation would measure. What the back-of-the-envelope establishes is the *direction*: for a memory-bound, data-movement-dominated workload that is only growing, the right architectural move is to make memory the center and bring compute to it. Everything in Section 5 is downstream of that conclusion.

7 Comparison and Positioning

To locate the NOETON precisely, we compare it along six axes against four reference points: the classical von Neumann machine, the conventional GPU+CPU server that trains and serves most models today, the AI PC of Section 3, and neuromorphic architectures, which share the NOETON's memory–compute fusion but pursue a different workload. Table 1 summarizes.

Three points stand out from the table. First, the NOETON is *not* the only architecture to fuse memory and compute—neuromorphic chips do so too [42, 46, 47]—but it does so for a different workload. Neuromorphic designs target sparse, event-driven spiking computation and excel at low-power sensing; the NOETON targets dense foundation-model inference. They share the diagnosis (von Neumann data movement is the enemy) but not the prescription. We see them as cousins, and a NOETON could plausibly host a neuromorphic island for always-on sensing within its degraded-mode floor.

Second, the GPU+CPU server already lives partway toward the NOETON: it is bandwidth-bound, it uses HBM to fight that, and large deployments disaggregate memory and split inference phases [22, 32]. The NOETON is in part an argument that these datacenter-scale adaptations—memory disaggregation, phase splitting, PIM offload—are not merely serving optimizations but previews of the *architecture*, and should be pushed down into the definition of the machine rather than bolted onto a fundamentally von Neumann server.

Table 1: The NOETON positioned against four reference architectures. The NOETON is distinguished not by any single column but by adopting the memory-centric, model-as-kernel, network-mandatory combination as its *organizing* commitment rather than as a bolt-on.

	von Neumann	**GPU+CPU server**	**AI PC**	**Neuromorphic**	**Noeton (ours)**
Memory model	Passive store, single bus	Deep cache hierarchy + HBM	Same + NPU scratchpad	Memory fused into neurons	Memory-centric: PIM + pooled/disaggregated weights
Compute paradigm	Sequential scalar core	SIMT throughput + scalar host	CPU + GPU + NPU device	Event-driven spiking	Tensor/attention dataflow; CPU demoted to housekeeping
"Operating system"	Hand-written executive	OS on CPU; model is a service	OS on CPU; model accelerated	N/A (fixed-function)	Model-OS: LLM is the kernel, agents are processes
Network assumption	Optional	Optional (cluster fabric)	Optional escalation to cloud	Typically standalone	**Mandatory**; offline is a degraded mode
Target workload	General sequential code	Training + batched inference	On-device small-model inference	Sparse spiking / sensing	Foundation-model inference & agents
Binding resource	Memory latency	Memory bandwidth	Memory bandwidth (inherited)	Energy per spike	Bandwidth—attacked at the source by PIM

Third, the AI PC differs from the NOETON in every row except "target workload," and that is the point of Section 3: it aims at the right workload with the wrong organization, inheriting the binding resource (bandwidth) it should have removed.

7.1 Operating-system correspondence

Because principle P3 (model-as-OS) is the layer most likely to be misread as metaphor, we make it concrete. Table 2 maps classical operating-system concepts to their Model-OS counterparts; each row corresponds to an existing line of systems work, cited in the table. The mapping is not an analogy for color—it is a specification of what the Model-OS kernel must provide and, by extension, what hardware support (protection domains over context and tool capabilities, fast context paging) the lower layers must offer.

8 Implications, Risks, and Open Challenges

A vision paper that only sells its vision is propaganda. The NOETON makes a genuinely costly choice—mandatory connectivity—and several hard problems are unsolved in the design above. We confront them here, because they are the strongest arguments against the proposal and the most important directions for work on it.

8.1 The cost of mandatory connectivity

Principle P4 deliberately makes the machine dependent on the network for full capability. This buys the central technical benefits—a small device presenting the interface of a very large model,

Table 2: Classical OS concepts and their Model-OS counterparts (P3). Each row names a concrete systems-research instantiation. The mapping doubles as a hardware requirements list: the kernel needs fast context paging and protection domains defined over context and tool capabilities.

Classical OS	**Model-OS counterpart**	**Instantiation / reference**
Kernel / privileged executive	The LLM itself	LLM OS framing [28]
Process	Agent	Agents-as-apps [14, 30]
System call	Tool / function invocation	Tool use [30]
Virtual memory / paging	Context-window management	MemGPT context paging [29]
File system	Vector store / retrieval index	Retrieval as addressing [28]
Scheduler	Agent/Task orchestrator	AIOS kernel scheduling [30]
Device driver	Tool adapter / connector	Tool adapters [14]
Memory protection	Context & capability isolation	Open problem (Section 8)

served from pooled memory it could never hold locally—but it sells real things, and we should name them rather than let the abstract win the argument by omission.

Privacy and surveillance. If full cognition happens over the network, then the user's intents—which on a cognitive machine are far more revealing than the files on a disk—traverse and may be processed by remote infrastructure. This is a structural increase in the surveillance surface relative to a self-contained PC. The honest response is not to deny it but to design against it: confidential-computing enclaves for inference, client-side context encryption, on-device handling of sensitive intents within the degraded-mode floor, and verifiable deletion. Apple's private-cloud-compute direction is an early attempt at exactly this trust structure [7]; whether such guarantees can be made strong and auditable enough to justify P4 is, in our view, the single most important open question for the whole proposal.

Autonomy and the "brick when offline" problem. A machine that does little when disconnected is a machine whose owner is less autonomous. We argued in Section 5.4 that the AI PC's optional-escalation model merely *hides* this dependency—its most useful capabilities already degrade or vanish offline—and that the Noeton at least makes the dependency explicit and designs a guaranteed floor (the on-device small model). But explicit is not the same as acceptable. The size of the degraded-mode floor is a policy choice with political content: how capable must a Noeton be offline before the dependency is tolerable? We do not think there is a single right answer, and we think the question deserves to be asked openly rather than settled silently by vendors.

Centralization and lock-in. If cognition is served from pooled weights in the network, whoever owns the pools owns a chokepoint over what the machine can think. This is a far more concentrated dependency than the application-store dynamics of today, and it invites the same failure modes—rent extraction, deplatforming, capricious policy—applied to cognition itself. Mitigations are partly technical (open weight pools, federated and peer-to-peer serving, the right to run your own tier) and partly governance, and we do not pretend the technical ones suffice on their own.

Equity and the digital divide. A machine whose full capability requires good connectivity widens the gap between the well-connected and everyone else, and exports it: the same device is a powerful cognitive tool in a city with fiber and a limited one in a region without it. The

degraded-mode floor is, again, the relevant lever—the more capable the offline small model, the smaller the divide—which is a reason to treat on-device capability as a floor to be raised, not a stopgap to be tolerated.

8.2 Unsolved technical problems

Beyond connectivity, the architecture leaves real problems open.

Protection over context and capabilities. Classical machines protect memory pages between processes. A Model-OS must protect *context and tool capabilities* between agent-processes—preventing one agent from reading another's context or invoking tools it should not—and we do not know the right hardware primitive for this (Table 2, last row). It is the model-as-kernel analogue of the memory-protection unit, and it does not yet exist.

Determinism, safety, and a learned kernel. A kernel that is a learned model is non-deterministic and can fail in ways a hand-written executive cannot: it can be prompt-injected, it can hallucinate a tool call, it can be jailbroken. Making a learned kernel trustworthy enough to hold privilege—sandboxing its tool calls, bounding its authority, verifying its actions—is a security agenda in itself, and the deterministic-dataflow choice of Layer 2 [25] addresses only the timing facet of it.

Programming and debugging an intent machine. When the program is weights and the instruction stream is tokens (P5), the classical tools of correctness—source, breakpoints, deterministic replay—do not straightforwardly apply. What does it mean to debug a NOETON? We raise the question without resolving it; it is plausibly as deep a shift as P1.

Software ecosystem and inertia. Finally, the largest practical obstacle is not silicon but the installed base. The von Neumann contract is the substrate of essentially all existing software. A NOETON must either emulate it (the demoted CPU of P2 is partly there for this) or coexist with it for a long transition. We expect the realistic path is hybrid—NOETON-like subsystems growing inside otherwise-conventional machines, exactly as the datacenter adaptations of Section 7 are already doing—rather than a clean replacement. The name is a destination, not a prediction of a flag day.

9 On Naming: Why Not Call It a Computer

We have used a new word throughout this paper, and the choice is not cosmetic. A name is a claim about what kind of thing something is, and we want to defend the claim that the machine we describe is different enough in kind to deserve one.

9.1 The argument for a new name

"Computer" is, etymologically and historically, a machine that *computes*—that carries out arithmetic and logical operations on data under the direction of a stored program. The word names the activity at the machine's center. By that test, the artifact of Section 5 is mis-named. Its central scarce resource is memory, not compute (P1); its dominant fabric runs a learned function rather than a stored program of hand-written instructions (P2, P3); its native unit of work is an intent resolved against context, not an instruction applied to addressed data (P5); and it is constitutively networked rather than self-contained (P4). When the center of the machine is no longer "compute" in the stored-program sense, continuing to call it a "computer" is a category error that quietly smuggles

the old contract back in—which is, we think, part of why the AI PC was conceived as an accelerated computer rather than as a different machine.

The historical precedent is real. "von Neumann machine" is itself a name that did useful work: it let architects reason about a *kind* of machine and its characteristic bottleneck [10, 15], rather than about any particular product. We want a name that does the same work for the AI era, standing in the relation

$$\text{von Neumann machine} \longrightarrow \textsc{Noeton},$$

so that one can say "this is a NOETON-style design" and thereby invoke P1–P5 as a set, the way "this is a von Neumann design" invokes fetch–decode–execute over a single store.

9.2 The proposed name

We propose **Noeton**, from the Greek *noesis* (), the act of understanding or intellection. The name foregrounds the shift from *computation* to *cognition* as the machine's organizing activity, parallels the personal honorific of "von Neumann machine" with a conceptual root instead, and is short, pronounceable, and unclaimed in the architecture literature. The architectural discipline—the set of principles a NOETON instantiates—we call the *Model-Native Architecture* (MNA), which travels well as an adjective ("a model-native design") independent of the proper noun.

On the Chinese rendering. The natural transliteration collides with the established Chinese name for Norton (the antivirus and the publisher), so we adopt as the recommended Chinese name: it preserves the *noe-* sound while the character ("to think") carries the same cognition-over-computation meaning the English coinage intends. We use NOETON in English and in Chinese throughout.

9.3 Alternatives we considered

We considered and rejected several alternatives, and record them because the choice is genuinely arguable. *Cognitive Engine* () is transparent and needs no gloss, but "engine" connotes a component inside a computer rather than a replacement for it, which undersells the claim. *Model-Native Machine (MNM)* is maximally descriptive and we keep "model-native" as the name of the architecture, but as a machine name it is a mouthful and reads as jargon. *Inference Machine*, *Tensor Machine*, and *Cognitron* each over-commit to one layer (the compute fabric, or a specific device style) at the expense of the whole. We prefer a name rooted in the machine's defining *activity*—understanding—over one rooted in its mechanism, for the same reason "computer" beat "arithmetic engine": activity names age better than mechanism names. A reader who dislikes NOETON can substitute their own; the substantive claim is that the artifact is different in kind and deserves *a* name, not that it must be this one.

10 Related Work

The NOETON is a synthesis, and its novelty is in the assembly rather than in any single component. We organize related work by the layer it informs, and state in each case what we take from it and where we depart.

Domain-specific architecture as a program. Our framing is a direct descendant of Hennessy and Patterson's call for a new golden age of domain-specific architectures co-designed with their workload [2], motivated by the end of Dennard scaling and dark silicon [1]. Where that program argues for DSAs in general, we argue that the foundation model is now *the* domain worth designing a whole machine around, and we take the co-design further—past the accelerator and into the operating system and the interface.

Memory-centric hardware. The substrate of Section 5.1 stands on processing-in-memory [19–21], attention-specific PIM offload [34], analog in-memory computing [35, 36], CXL memory pooling and disaggregation [22, 23, 37], and wafer-scale integration [24, 38, 39], alongside software that already treats inference itself as a memory-hierarchy offloading problem across GPU, CPU, and disk [48]. Each of these treats memory-centricity as a technique to be applied within a conventional system; our departure is to treat it as the *organizing principle* of the machine, with the others arranged around it.

Tensor and dataflow accelerators. The compute fabric draws on systolic tensor engines [27], attention accelerators [40, 41], and deterministic dataflow processors [25, 26, 49]. These are typically positioned as accelerators *for* a host; we position the fabric as the primary compute of the machine and the host CPU as the accessory (P2).

The von Neumann critique and its alternatives. The structural critique we build on is old [10, 15], and neuromorphic computing is the most committed existing attempt to escape it [42, 46, 47, 50]. We share the diagnosis and differ in the target workload (Section 7): dense foundation-model inference rather than sparse spiking computation.

The model as operating system. Layer 3 adopts the rapidly developing "LLM OS" line—Karpathy's framing [28], MemGPT's OS-inspired context paging [29], and the AIOS kernel and AIOS-agent ecosystem [14, 30]. This work is almost entirely *software*, running on conventional hardware. Our contribution is to ask what hardware a model-as-kernel demands, and to make that demand the design driver for the layers beneath it. The closest precedent in genre is the AIOS-agent vision paper [14]; the closest in spirit but opposite in scope, since it envisions the software ecosystem while we re-architect the machine under it.

Networked and split inference. Layer 4 builds on phase-split serving [32], hybrid on-device/cloud foundation models [7, 31], and the long thin-client lineage descending from "the network is the computer" [33]. The systems work treats split inference as a serving optimization; we treat mandatory, tiered cognition as a defining property of the machine, with an explicit degraded mode.

Workload and economics. Finally, the motivation rests on the Transformer and the scale-driven character of its capability [3–5, 17] (with the strong reading of emergence contested [18]), on roofline analyses of LLM inference [9, 12, 13], on the data-movement energy gap [16], and on the rising energy cost of serving at scale [43–45]. These establish that the workload is large, growing, bandwidth-bound, and energy-intensive—the four facts from which the whole design follows.

11 Industry Trajectory (2024–2026): Evidence Since Drafting

A position paper should be falsifiable by the market it describes. In the window surrounding the drafting of this paper, the industry moved measurably toward each of the five principles; we record the clearest data points here, together with two refinements they force on our own claims.

Toward memory-centric organization (P1). The KVCache-centric serving system Mooncake—which reorganizes an entire production cluster around the cache rather than the compute—received the Best Paper Award at FAST 2025 and has since been integrated into the mainstream vLLM stack [51]. On the device side, the two leading DRAM makers began jointly standardizing LPDDR6-PIM [52], while CXL 3.1 memory-expansion controllers reached customer sampling. Most strikingly,

Huawei's CloudMatrix 384 supernode connects NPUs, CPUs, DRAM, and SSDs over a unified bus *without CPU mediation*, offering roughly 3.6× the aggregate memory capacity and 2.1× the bandwidth of a comparable GPU rack [53]: pooled, peer-addressed memory at rack scale is no longer hypothetical.

Toward intensity-matched compute (P2) and split inference (P4). The strongest single confirmation comes from the incumbent: NVIDIA's Rubin CPX is a GPU *built only for prefill*—high arithmetic throughput paired deliberately with commodity GDDR7 rather than HBM—while its HBM-equipped sibling serves the bandwidth-bound decode phase, and the Dynamo framework schedules requests across the two pools [54, 55]. When the vendor with the least incentive to fragment the GPU ships phase-specialized silicon, the claim that prefill and decode "do not belong on the same chip" has crossed from position to product.

Toward the Model-OS (P3) and intent interfaces (P5). The Model Context Protocol went from a single-vendor release (Nov. 2024) to adoption by every major model provider and then to neutral governance under the Linux Foundation within thirteen months [56]—the fastest standardization of a "system-call" surface in computing history, driven by the same $N \times M$ adapter explosion that once motivated POSIX. Apple's deployment of an on-device model backed by Private Cloud Compute [57] put a two-tier degraded-mode design, consistent with our tiered-cognition principle, into hundreds of millions of consumer devices.

Two refinements. The evidence also corrects us in two places. First, *near*-memory is arriving before *in*-memory: custom HBM base dies, CXL pooling, and supernode fabrics are commercializing faster than in-bank PIM, so the L1 substrate should be read as a progression (near-memory pooling → bank-level PIM) rather than a single step. Second, KV-cache compression (multi-head latent attention, sparse attention) moderates cache *capacity* growth; it does not, however, move decode off the bandwidth-bound slope of the roofline—if anything, test-time-compute workloads, which spend their tokens at decode, increase the share of inference that lives there.

None of these developments was coordinated with this paper; all of them bend in its direction. We take this as evidence that the von Neumann → NOETON reading of the present moment is, at minimum, a productive way to see where the industry is already going.

12 Conclusion

For seventy years we have built von Neumann machines and called them computers, because at their center was computation over a stored program. We have argued that the dominant workload of the emerging era—foundation-model inference—does not fit that center. It is bound by the movement of weights and cache, not by arithmetic; its natural kernel is a learned model, not a hand-written executive; its natural unit of work is an intent, not an instruction; and its full capability is, increasingly, something drawn from the network rather than held in a box. The industry's first response, the AI PC, accelerates the model within the old contract and inherits the very bottleneck it should remove.

We proposed to take the alternative seriously: to redesign the machine, from the hardware up, around the model. The result—the NOETON, organized by the Model-Native Architecture—rests on five principles (memory-centric organization, model-native compute, model-as-operating-system, network-mandatory tiered cognition, and intent-driven interaction) and assembles existing, demonstrated technologies—processing-in-memory, disaggregated memory, wafer-scale integration, deterministic dataflow fabrics, and the emerging LLM OS—into a single coherent machine that a

roofline and an energy ledger say is pointed the right way. We named it deliberately, because when the center of a machine is understanding rather than computation, the old name smuggles in the old assumptions.

We have been candid about what this costs. Mandatory connectivity trades autonomy, privacy, and equity for capability, and several of the architecture's hardest problems—protection over context and capabilities, the safety of a learned kernel, the meaning of debugging an intent machine—are open. We regard these not as objections that sink the proposal but as the research agenda the proposal generates.

A research agenda. If the direction is right, the concrete next questions are:

1. *Substrate.* What is the right balance, for a given device class, between resident (wafer-scale/HBM) and paged (CXL/network) weight memory, and what PIM-to-fabric interface does the operator-placement split of Section 5.2 require?

2. *Kernel.* What is the hardware protection primitive for context and tool capabilities (the missing row of Table 2), and how is a learned kernel bounded enough to safely hold privilege?

3. *Network.* How large must the degraded-mode floor be for the dependency of P4 to be acceptable, and can the privacy guarantees that would justify mandatory connectivity be made strong and auditable?

4. *Evaluation.* What does a fair benchmark of a NOETON against an AI PC even measure, when the two make different assumptions about where capability lives?

We do not expect a flag day. The realistic path is the one already visible in the datacenter, where memory disaggregation, phase splitting, and PIM offload are quietly turning servers into something more NOETON-like one optimization at a time. Our contribution is to name the destination those optimizations are walking toward, to argue it is a coherent place and not an accident, and to put the costs of arriving there on the table alongside the benefits. Whether or not the word NOETON survives, we believe the machine it names is worth building toward—and worth arguing about now, while its contract is still being written.

Acknowledgments

This is a vision paper; no new systems were built or evaluated. We thank the broader computer-architecture and machine-learning-systems communities, whose work we synthesize and on whose shoulders this proposal stands.

References

[1] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proc. 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 365–376, 2011. doi: 10.1145/2000064.2000108.

[2] John L. Hennessy and David A. Patterson. A new golden age for computer architecture. *Communications of the ACM*, 62(2):48–60, 2019. doi: 10.1145/3282307.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. arXiv:1706.03762.

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1877–1901, 2020. GPT-3; arXiv:2005.14165.

[5] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[6] Microsoft. Introducing Copilot+ PCs. Microsoft Official Blog, May 20, 2024, 2024. https://blogs.microsoft.com/blog/2024/05/20/introducing-copilot-pcs/.

[7] Apple Machine Learning Research. Introducing Apple's on-device and server foundation models, 2024. https://machinelearning.apple.com/research/introducing-apple-foundation-models.

[8] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

[9] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems (MLSys)*, 2023. arXiv:2211.05102.

[10] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, 1995. doi: 10.1145/216585.216588.

[11] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W. Mahoney, and Kurt Keutzer. AI and memory wall. *IEEE Micro*, 44(3):33–39, 2024. doi: 10.1109/MM.2024.3373763. arXiv:2403.14123.

[12] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009. doi: 10.1145/1498765.1498785.

[13] Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, et al. LLM inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*, 2024.

[14] Yingqiang Ge, Yujie Ren, Wenyue Hua, Shuyuan Xu, Juntao Tan, and Yongfeng Zhang. LLM as OS, agents as apps: Envisioning AIOS, agents and the AIOS-agent ecosystem. *arXiv preprint arXiv:2312.03815*, 2023.

[15] John Backus. Can programming be liberated from the von Neumann style? a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978. doi: 10.1145/359576.359579. 1977 ACM Turing Award Lecture.

[16] Mark Horowitz. 1.1 Computing's energy problem (and what we can do about it). In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 10–14, 2014. doi: 10.1109/ISSCC.2014.6757323.

[17] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, et al. Emergent abilities of large language models. *Transactions on Machine Learning Research (TMLR)*, 2022. arXiv:2206.07682.

[18] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2304.15004.

[19] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. A modern primer on processing in memory. *Emerging Computing: From Devices to Systems (Springer); arXiv:2012.03112*, 2022.

[20] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture. *IEEE Access*, 10:52565–52608, 2022. doi: 10.1109/ACCESS.2022.3174101. PrIM benchmarks, UPMEM; arXiv:2105.03814.

[21] Young-Cheon Kwon, Suk Han Lee, Jaehoon Lee, et al. Aquabolt-XL: Samsung HBM2-PIM with in-memory processing for ML accelerators and beyond. In *IEEE Hot Chips 33 Symposium (HCS)*, 2021. doi: 10.1109/HCS52781.2021.9567191.

[22] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, et al. Pond: CXL-based memory pooling systems for cloud platforms. In *Proc. 28th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023. doi: 10.1145/3575693.3578835. Distinguished Paper; arXiv:2203.00241.

[23] Hasan Al Maruf and Mosharaf Chowdhury. Memory disaggregation: Advances and open challenges. *ACM SIGOPS Operating Systems Review*, 57(1), 2023. doi: 10.1145/3606557.3606562. arXiv:2305.03943.

[24] Sean Lie. Cerebras architecture deep dive: First look inside the hardware/software co-design for deep learning. *IEEE Micro*, 43(3):18–30, 2023. doi: 10.1109/MM.2023.3256384.

[25] Dennis Abts, Jonathan Ross, Jonathan Sliwa, Brian Taylor, Chia-Hsin Owen Chen, Gregory Thorson, Igor Tomic, et al. Think fast: A tensor streaming processor (tsp) for accelerating deep learning workloads. In *Proc. 47th Annual Int. Symposium on Computer Architecture (ISCA)*, pages 145–158, 2020. doi: 10.1109/ISCA45697.2020.00023.

[26] Raghu Prabhakar, Ram Sivaramakrishnan, Darshan Gandhi, Yun Du, Mingran Wang, Xiangyu Song, et al. SambaNova SN40L: Scaling the AI memory wall with dataflow and composition of experts. *Proc. 57th IEEE/ACM Int. Symposium on Microarchitecture (MICRO); arXiv:2405.07518*, 2024.

[27] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. 44th Annual Int. Symposium on Computer Architecture (ISCA)*, pages 1–12, 2017. doi: 10.1145/3079856.3080246. arXiv:1704.04760.

[28] Andrej Karpathy. LLM OS: The LLM as the kernel process of a new operating system. https://x.com/karpathy/status/1707437820045062561, 2023. Talk "Intro to Large Language Models" and accompanying remarks; not peer-reviewed.

[29] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[30] Kai Mei, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. AIOS: LLM agent operating system. *arXiv preprint arXiv:2403.16971*, 2024.

[31] Apple. Apple intelligence foundation language models. *arXiv preprint arXiv:2507.13575*, 2025.

[32] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *Proc. 51st Annual Int. Symposium on Computer Architecture (ISCA)*, 2024. doi: 10.1109/ISCA59077.2024.00019. arXiv:2311.18677.

[33] John Gage. The network is the computer, 1984. Sun Microsystems slogan; see IEEE Spectrum coverage, https://spectrum.ieee.org/does-repurposing-of-sun-microsystems-slogan-honor-history.

[34] Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk Kwon, Nam Sung Kim, and Jung Ho Ahn. AttAcc! unleashing the power of PIM for batched transformer-based generative model inference. In *Proc. 29th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024. doi: 10.1145/3620665.3640422.

[35] Manuel Le Gallo, Riduan Khaddam-Aljameh, Milos Stanisavljevic, et al. Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators. *Nature Communications*, 14:5282, 2023. doi: 10.1038/s41467-023-40770-4.

[36] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *Proc. 43rd Int. Symposium on Computer Architecture (ISCA)*, pages 14–26, 2016. doi: 10.1145/3007787.3001139.

[37] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. TPP: Transparent page placement for CXL-enabled tiered-memory. In *Proc. 28th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023. doi: 10.1145/3582016.3582063. arXiv:2206.02878.

[38] Sean Lie. Inside the cerebras wafer-scale cluster. *IEEE Micro*, 44(4), 2024. doi: 10.1109/MM.2024.3386628.

[39] Nolan Dey et al. Benchmarking the performance of large language models on the cerebras wafer-scale engine. *arXiv preprint arXiv:2409.00287*, 2024.

[40] Tae Jun Ham, Sung Jun Jung, Seonghak Kim, Young H. Oh, Yeonhong Park, Yoonho Song, Jung-Hun Park, Sanghee Lee, Kyoung Park, et al. A^3: Accelerating attention mechanisms in neural networks with approximation. In *IEEE Int. Symposium on High Performance Computer Architecture (HPCA)*, pages 328–341, 2020. doi: 10.1109/HPCA47549.2020.00035. arXiv:2002.10941.

[41] Hanrui Wang, Zhekai Zhang, and Song Han. SpAtten: Efficient sparse attention architecture with cascade token and head pruning. In *IEEE Int. Symposium on High Performance Computer Architecture (HPCA)*, pages 97–110, 2021. doi: 10.1109/HPCA51647.2021.00018. arXiv:2012.09852.

[42] Dharmendra S. Modha, Filipp Akopyan, Alexander Andreopoulos, Rathinakumar Appuswamy, John V. Arthur, et al. Neural inference at the frontier of energy, space, and time. *Science*, 382 (6668):329–335, 2023. doi: 10.1126/science.adh1174. IBM NorthPole.

[43] Anonymous. How hungry is AI? benchmarking energy, water, and carbon footprint of LLM inference. *arXiv preprint arXiv:2505.09598*, 2025.

[44] Anonymous. Quantifying the energy consumption and carbon emissions of LLM inference via simulations. *arXiv preprint arXiv:2507.11417*, 2025.

[45] Arman Shehabi, Sarah J. Smith, Alex Hubbard, Alexander Newkirk, Nuoa Lei, Md Abu Bakar Siddik, Billie Holecek, Jonathan Koomey, Eric Masanet, and Dale Sartor. 2024 united states data center energy usage report. Technical Report LBNL-2001637, Lawrence Berkeley National Laboratory, 2024.

[46] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014. doi: 10.1126/science.1254642.

[47] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018. doi: 10.1109/MM.2018.112130359.

[48] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E. Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proc. 40th Int. Conference on Machine Learning (ICML)*, 2023. arXiv:2303.06865.

[49] Zhe Jia, Blake Tillman, Marco Maggioni, and Daniele P. Scarpazza. Dissecting the graphcore IPU architecture via microbenchmarking. *arXiv preprint arXiv:1912.03413*, 2019.

[50] Steve B. Furber, Francesco Galluppi, Steve Temple, and Luis A. Plana. The SpiNNaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014. doi: 10.1109/JPROC.2014.2304638.

[51] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *USENIX Conference on File and Storage Technologies (FAST)*, 2025. Best Paper Award; arXiv:2407.00079.

[52] SK hynix and Samsung Electronics. LPDDR6-PIM standardization initiative for low-power processing-in-memory, 2025. Industry roadmap disclosures, 2025.

[53] Pengfei Zuo et al. Serving large language models on Huawei CloudMatrix384, 2025. arXiv:2506.12708.

[54] NVIDIA. NVIDIA unveils Rubin CPX: A new class of GPU designed for massive-context inference, September 2025. Press release. https://nvidianews.nvidia.com/news/nvidia-unveils-rubin-cpx-a-new-class-of-gpu-designed-for-massive-context-inference.

[55] NVIDIA. NVIDIA Dynamo: A datacenter-scale distributed inference serving framework, March 2025. Announced at GTC 2025. https://github.com/ai-dynamo/dynamo.

[56] Anthropic. Introducing the Model Context Protocol, November 2024. Donated to the Linux Foundation Agentic AI Foundation in Dec. 2025. https://modelcontextprotocol.io.

[57] Apple Security Engineering. Private cloud compute: A new frontier for AI privacy in the cloud, June 2024. https://security.apple.com/blog/private-cloud-compute/.