SvedbergOpen
DISSEMINATIONOFKNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

The Era of Beyond-von Neumann Computation from Cache-Aware Multicore Systems to Memory-Centric, Neuromorphic, Photonic, and Quantum Machines

[1]Goutam Mondal, [2]Purnendu Das, [3,*]Bishwa Ranjan Roy, [4]Nurulla Mansur Barbhuiya

[1,2,3,4]Department of Computer Science, Assam University, Silchar, India
*E-mail: brroy88@gmail.com

Abstract
The von Neumann model remains the foundation of general-purpose computing, yet its separation of processing and memory increasingly exposes data movement, energy, and scalability limits. Modern multicore processors respond with deep cache hierarchies, prefetchers, coherence protocols, and high-bandwidth memory. These mechanisms are essential, but they mainly hide or amortize the cost of moving operands rather than remove it. This review connects that conventional baseline to the emerging era of beyond-von Neumann computation. It organizes the field into near- and in-memory computing, neuromorphic systems, dataflow and spatial architectures, photonic computing, quantum accelerators, and approximate or probabilistic methods. For each paradigm, the paper explains the execution model, representative hardware, application affinity, advantages, and unresolved barriers. Cross-paradigm tables and an author-synthesized design-space map show that no single architecture dominates across programmability, precision, data-movement reduction, and maturity. The most plausible trajectory is therefore heterogeneous: conventional CPUs orchestrate specialized engines connected through common memory, packaging, compiler, and runtime abstractions. The review concludes with research priorities for end-to-end energy accounting, reliability, security, benchmark methodology, and software portability.

Index Terms - beyond-von Neumann computing, compute-in-memory, prefetching, neuromorphic computing, dataflow architecture, photonic computing, quantum computing, heterogeneous acceleration.

I. Introduction

The stored-program computer described in the EDVAC report established a remarkably durable contract: instructions and data occupy an addressable memory, while a processor repeatedly fetches, decodes, and executes them [1]. The abstraction enabled software portability and decades of microarchitectural innovation. Its central weakness is equally durable. Computation and state are physically separated, so useful work is accompanied by communication across registers, caches, interconnects, and off-chip memory. Wulf and McKee described the resulting processor-memory imbalance as the memory wall [2]. Multicore scaling later amplified the problem because more cores compete for shared capacity and bandwidth, while coherence and synchronization add further traffic [3].

Conventional processors have not stood still. Out-of-order execution overlaps independent misses; private caches preserve locality; shared last-level caches reduce off-chip requests; prefetchers predict future references; and chiplets or stacked memory widen the data path. Such techniques remain indispensable. However, their success can obscure the deeper issue: the energy and latency of transporting an operand may exceed the cost of the arithmetic that consumes it. The renewed emphasis on domain-specific architecture therefore reflects both the end of easy transistor-level scaling and the rise of workloads dominated by large, reusable data structures [4].

This paper uses the conventional multicore hierarchy as a point of departure rather than a straw man. It first explains why cache memories and prefetching are effective, then identifies the conditions under which their benefits saturate. It subsequently reviews six families of beyond-von Neumann methods. The phrase is used broadly but carefully: some methods move computation toward memory, some replace instruction-centric control with dataflow, and others exploit nontraditional devices or physics. The goal is not to claim that every accelerator abandons the stored-program model, but to compare how strongly each one relaxes its assumptions. Recent continuum-level reviews similarly emphasize heterogeneous combinations rather than a single replacement [23].

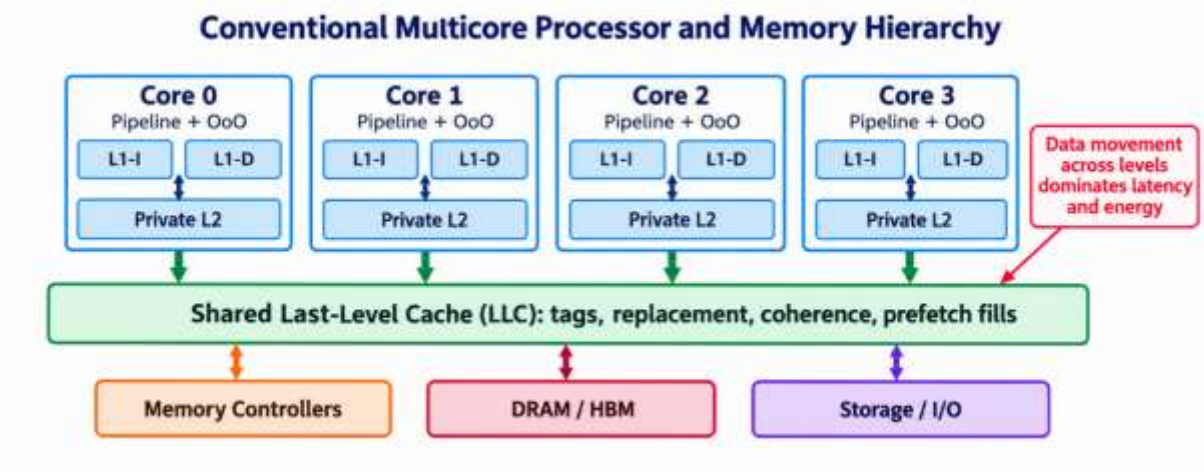


Fig. 1. Conventional multicore processor and cache-memory hierarchy.

II. THE CONVENTIONAL BASELINE: CACHES, MULTICORES, AND PREFETCHING

A. Multicore and Cache-Memory Structure

Figure 1 shows a representative chip multiprocessor. Each core combines a pipeline and out-of-order engine with small private instruction and data caches, commonly followed by a private mid-level cache. A larger shared last-level cache (LLC) acts as a capacity reservoir and coherence point before requests reach memory controllers and DRAM or high-bandwidth memory. This hierarchy exploits temporal and spatial locality: recently accessed blocks are retained near the core, and transfers occur in cache-line granularity to amortize tags, wires, and DRAM activation.

The hierarchy is highly effective when a working set fits, when reuse occurs before eviction, and when cores share data predictably. Its limitations appear when capacity misses dominate, address streams are irregular, or many cores generate concurrent misses. Larger caches also incur longer access time and leakage; aggressive sharing introduces interference; and coherence can invalidate data that was prefetched or replicated. Thus, cache design is a continuing optimization of latency, capacity, bandwidth, isolation, and energy rather than a complete resolution of the memory wall.

B. Why Prefetching Exists

A demand miss exposes the critical path of lower-level cache or memory access. Hardware or software prefetching attempts to issue that request earlier, ideally placing the block in an appropriate cache shortly before it is consumed. Jouppi's early prefetch buffers demonstrated how sequential streams could be decoupled from the demand path [5]; subsequent designs learned strides, spatial footprints, temporal correlations, and context-sensitive patterns. A useful prefetcher must balance three related properties: coverage, the fraction of harmful misses it predicts; accuracy, the fraction of its requests that are actually used; and timeliness, whether useful data arrives before the demand.

Figure 2 illustrates the control loop. A demand stream feeds a pattern detector, whose confidence and throttling logic decides whether and where to issue a prefetch. A timely and accurate request hides latency and may improve row-buffer locality. A late request consumes bandwidth without eliminating the stall, while an inaccurate request can evict useful lines, increase DRAM traffic and energy, and interfere with other cores. Server studies consequently show that no single hardware prefetcher is universally best and that system conditions strongly influence its value [6]. Prefetching is also less effective for dependent pointer chains, rapidly changing graph frontiers, sparse accesses with weak history, encrypted or randomized address mappings, and workloads whose working set exceeds sustainable bandwidth.

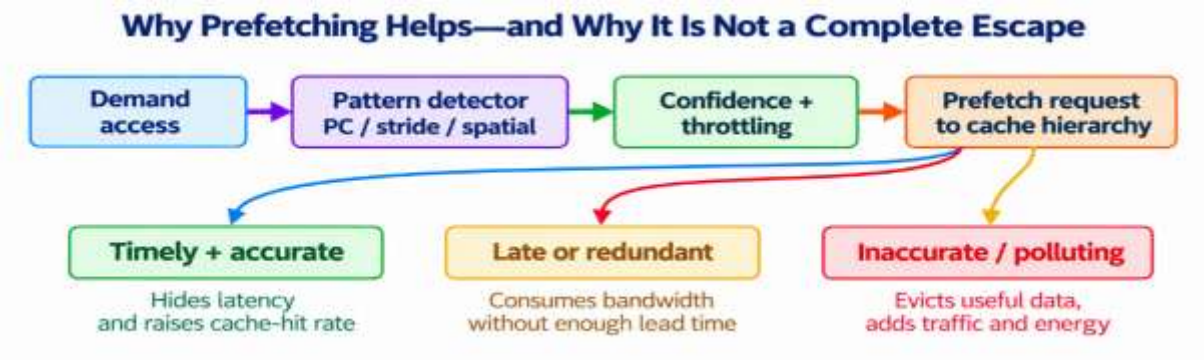


Fig. 2. Prefetching predicts future accesses, but usefulness depends on accuracy and timeliness.

The key distinction is conceptual. Caching relies on observed locality; prefetching speculates about future locality. Both preserve the processor-memory separation. They reduce the exposed penalty of moving data but cannot make movement free. Beyond-von Neumann designs become attractive when a workload repeatedly moves large volumes of data to perform relatively simple operations, when sparse events make clocked execution wasteful, or when a physical process can directly realize a useful mathematical primitive.

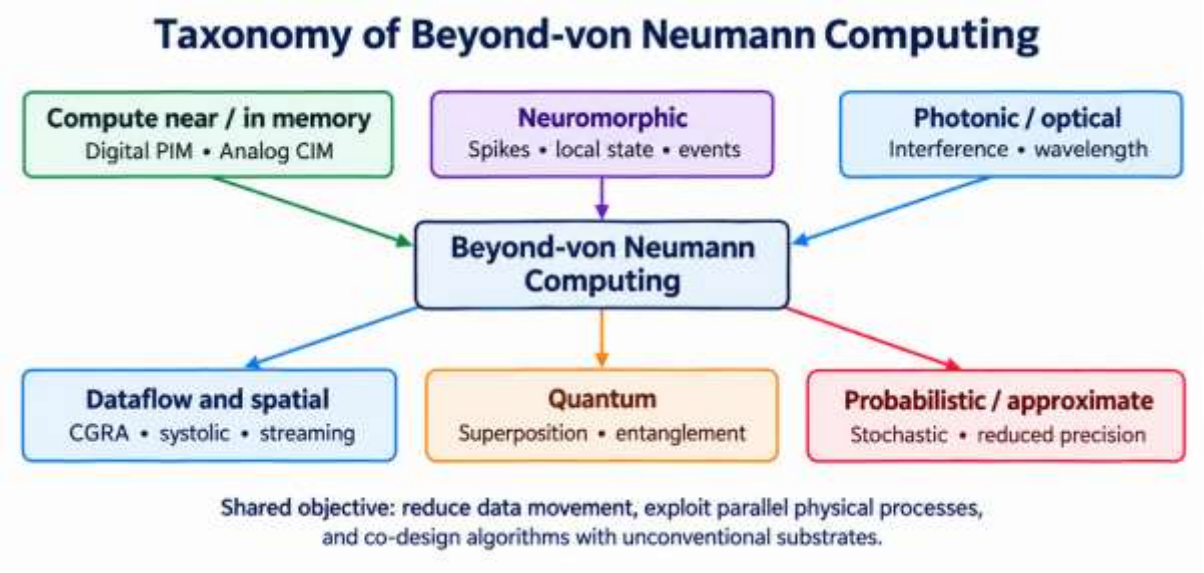


Fig. 3. Taxonomy of beyond-von Neumann methods considered in this review.

TABLE I FROM LATENCY HIDING TO ARCHITECTURAL DEPARTURE

Method	Primary bottleneck attacked	Core idea	What remains expensive
Caches	Average access latency	Retain recently used blocks near cores	Misses, capacity, coherence, leakage
Prefetching	Exposed miss latency	Predict and fetch blocks before demand	Bandwidth, pollution, irregularity
Near-memory	Off-chip movement	Place programmable logic near stacked memory	Movement inside memory stack, mapping
Compute-in-memory	Repeated operand transport	Operate where weights/data are stored	Conversion, precision, endurance
Neuromorphic	Dense clocked processing of sparse signals	Event-driven neurons with local state	Training, programmability, accuracy
Dataflow/ spatial	Instruction/control overhead and locality loss	Fire on readiness or spatially map operations	Compilation, dynamic behavior
Photonic	Bandwidth and linear algebra throughput	Use wavelength/interference for communication or MVM	ADC/DAC, nonlinearity, calibration
Quantum	Selected combinatorial/quantum-state problems	Use superposition and entanglement	Errors, qubit overhead, classical control

III. COMPUTE NEAR OR INSIDE MEMORY

A. Near-Memory and Processing-in-Memory

Near-memory processing places compute units in the logic layer of a 3-D memory stack, beside memory controllers, or within memory modules. The arithmetic is usually digital and may remain programmable, but it accesses high internal bandwidth without repeatedly traversing package and board interfaces. This

approach is especially attractive for database scans, graph analytics, vector operations, and reductions whose arithmetic intensity is too low to justify round trips to a CPU. The main architectural questions are which operations to offload, how virtual memory and coherence are maintained, and whether data placement exposes enough parallelism to amortize invocation overhead.

Processing-in-memory is sometimes used as an umbrella term, but distinguishing near-memory logic from operations performed by memory arrays is useful. Near-memory engines preserve conventional digital correctness and are comparatively easy to integrate, yet they do not eliminate all data movement. Array-level compute can reduce movement more aggressively, but it couples algorithms to device behavior and peripheral circuitry. Recent surveys accordingly organize designs from circuit primitives through processor interfaces and applications [7], [8].

B. Digital and Analog Compute-in-Memory

Digital compute-in-memory (CIM) augments SRAM, DRAM, or nonvolatile arrays with logic for bitwise operations, comparisons, additions, or reductions. It can support exact arithmetic and familiar verification, but additional wordline control, sense amplifiers, and data-layout constraints limit flexibility. Analog CIM uses device conductance to encode matrix weights. Input voltages applied along rows create column currents, so Ohm's and Kirchhoff's laws evaluate many multiply-accumulate operations in parallel as shown in figure 4. This makes matrix-vector multiplication a physical primitive rather than a sequence of fetched instructions.

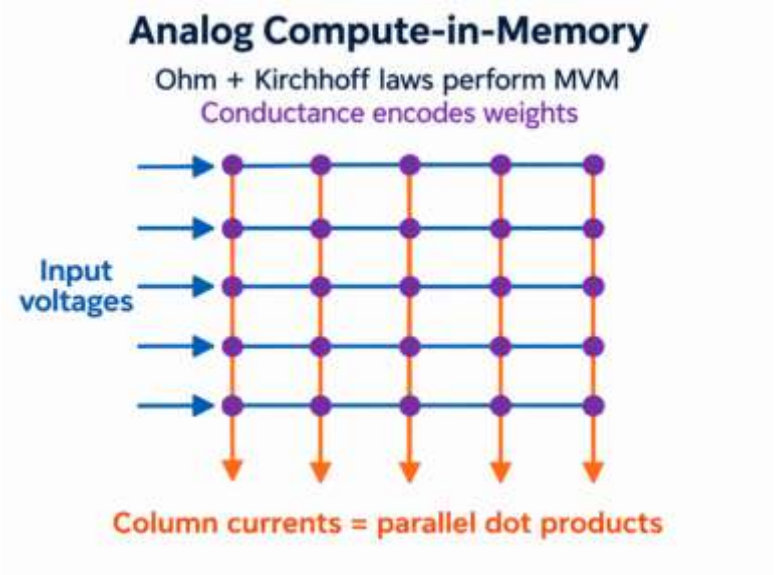


Fig. 4. Analog compute-in-memory crossbar for matrix-vector multiplication.

The apparent efficiency of an array is not the same as end-to-end system efficiency. Digital-to-analog converters, analog-to-digital converters, accumulation, activation functions, calibration, data reordering, and host transfers can dominate. Device variation, drift, limited conductance levels, write endurance, and temperature perturb the computation [9]. Mixed-precision schemes therefore combine low-precision analog operations with accurate digital correction [25], while device-aware training and analog-aware hardware models absorb systematic errors [26]. Demonstrations of analog-memory neural-network training show that equivalent application accuracy is possible, but only through co-design of algorithms, circuits, and update rules [10].

IV. NEUROMORPHIC COMPUTING

Neuromorphic systems take inspiration from biological neural organization rather than merely accelerating conventional artificial neural networks as shown in figure 5. State is distributed across neuron and synapse circuits; communication uses sparse spikes; and work is triggered by events instead of a global sequence of dense clocked operations. Co-locating synaptic state with neuron updates reduces repeated weight movement, while asynchronous activity allows quiescent regions to consume little dynamic energy.

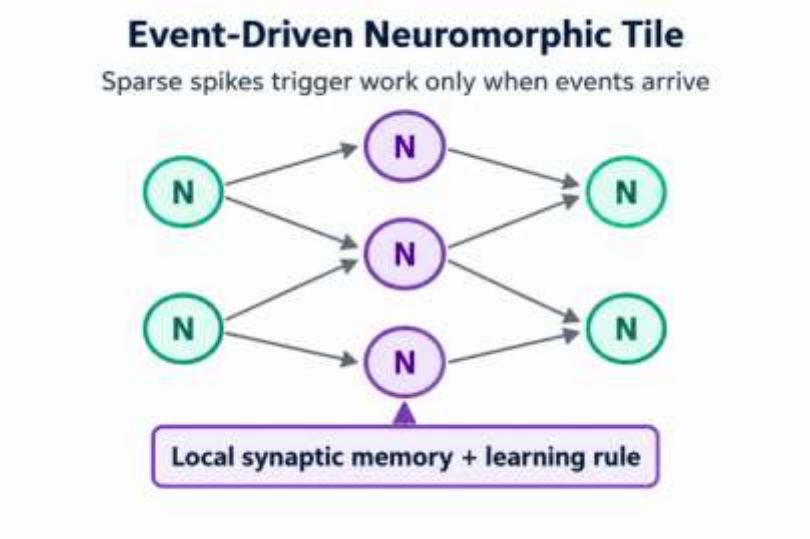


Fig. 5. Event-driven neuromorphic tile with local synaptic state.

IBM TrueNorth integrated one million programmable spiking neurons with a scalable communication fabric [11]. Intel Loihi added programmable on-chip learning and a manycore mesh, making adaptation part of the architecture rather than an offline host function [12]. Such chips are compelling for always-on sensing, temporal pattern recognition, control, event cameras, and edge inference, particularly when input sparsity and latency matter more than dense floating-point throughput. At scale, however, routing spikes, partitioning graphs, and preserving biological or algorithmic timing semantics become first-class concerns [13], [27].

The software gap is presently more limiting than neuron count. Spiking neural networks have discontinuous event dynamics, and converting a high-accuracy dense model can erase latency or energy advantages. Direct training is improving, but frameworks, datasets, and evaluation practices remain fragmented. Comparisons must include encoding and sensor costs, time-to-solution, accuracy, and idle activity rather than report only energy per synaptic operation. Neuromorphic computing is therefore best viewed as a distinct event-processing substrate with strong application affinity, not as a universal low-power replacement for GPUs.

V. DATAFLOW, SPATIAL, AND DOMAIN-SPECIFIC ARCHITECTURES

A conventional processor names a dynamic instruction sequence and devotes substantial hardware to discovering parallelism. As shown in Figure 6, Dataflow architectures invert this relationship: an operation becomes eligible when its operands are available. WaveScalar demonstrated a scalable dataflow instruction set while retaining a memory ordering mechanism for imperative programs [14]. Spatial architectures and coarse-grained reconfigurable arrays (CGRAs) take a related approach by mapping operations and communication paths onto a fabric. Values can flow directly between functional units, reducing register-file traffic and control overhead.

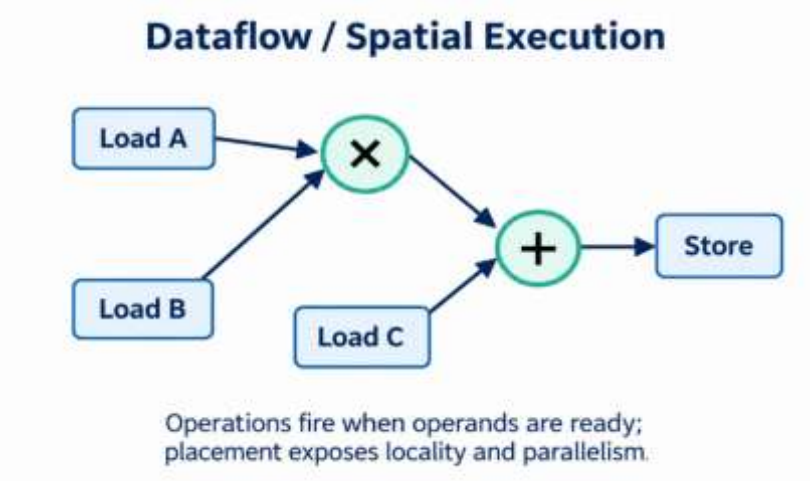


Fig. 6. Dataflow/spatial execution exposes producer-consumer locality.

The approach is especially effective for regular pipelines, signal processing, tensor kernels, stencil computations, and streaming graphs. Compilation determines placement, routing, buffering, and initiation interval, so the compiler becomes part of the microarchitecture. CGRA surveys show a wide design space spanning reconfiguration granularity, memory organization, and static versus dynamic scheduling [15]. Dynamic branches, pointer-rich code, irregular memory dependence, and rapid phase changes remain difficult because remapping can cost more than the saved execution energy.

Domain-specific accelerators occupy an important middle ground. Google's tensor processing unit retains a conventional host but replaces general-purpose instruction issue with a systolic matrix engine and explicitly managed memory, achieving large gains for production neural networks [16], [24]. Such systems may not be beyond-von Neumann in a strict physical sense, yet they depart from general-purpose control and demonstrate the practical power of matching dataflow and storage to a domain. Their risk is brittleness: an accelerator can become underutilized when model structure, precision, or operator mix changes.

VI. PHOTONIC AND OPTICAL COMPUTING

Photonic computing uses light for communication and, in some designs, for arithmetic. Wavelength-division multiplexing allows multiple channels to share a waveguide, while interferometers and resonators implement weighted linear transforms. In Figure 7, Coherent nanophotonic circuits have demonstrated neural-network inference in which interference performs matrix operations at the speed of optical propagation [17]. Optical links can also relieve electrical interconnect energy and bandwidth limits without changing the compute model.

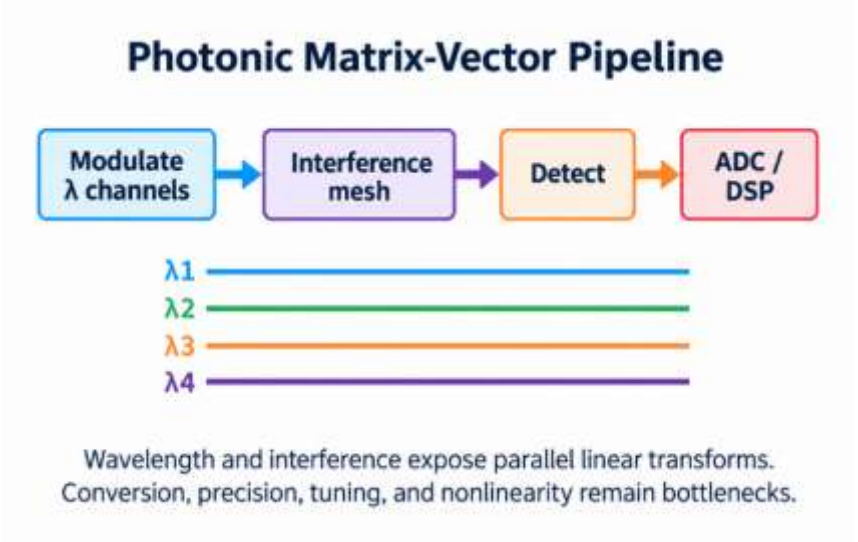


Fig. 7. Photonic matrix-vector pipeline and its electronic boundaries.

The most favorable operation is a large, reused linear transform. A photonic core can expose enormous parallelism, but a complete neural layer also needs memory, modulation, photodetection, nonlinear activation, control, and often ADC/DAC conversion. Laser efficiency, thermal tuning, phase noise, component variation, insertion loss, and finite precision determine system behavior. Recent reviews therefore emphasize integrated sources, detectors, calibration, and electronic-photonic co-design rather than isolated optical throughput [18], [19]. Photonics is likely to enter systems incrementally-first as interconnect and specialized linear algebra-rather than replace general-purpose electronic logic.

VII. QUANTUM COMPUTING AS A SPECIALIZED ACCELERATOR

Quantum computing changes the state space itself. Qubits support superposition, entanglement, and interference; algorithms manipulate amplitudes so that measurement favors useful outcomes. This does not imply exponential speedup for ordinary code. Advantage depends on a problem having a suitable quantum algorithm, on data being prepared without overwhelming cost, and on the circuit completing with acceptable error. The current noisy intermediate-scale quantum (NISQ) regime therefore relies on hybrid loops in which a classical host prepares circuits, optimizes parameters, performs mitigation, and interprets samples [20] as shown in Figure 8.

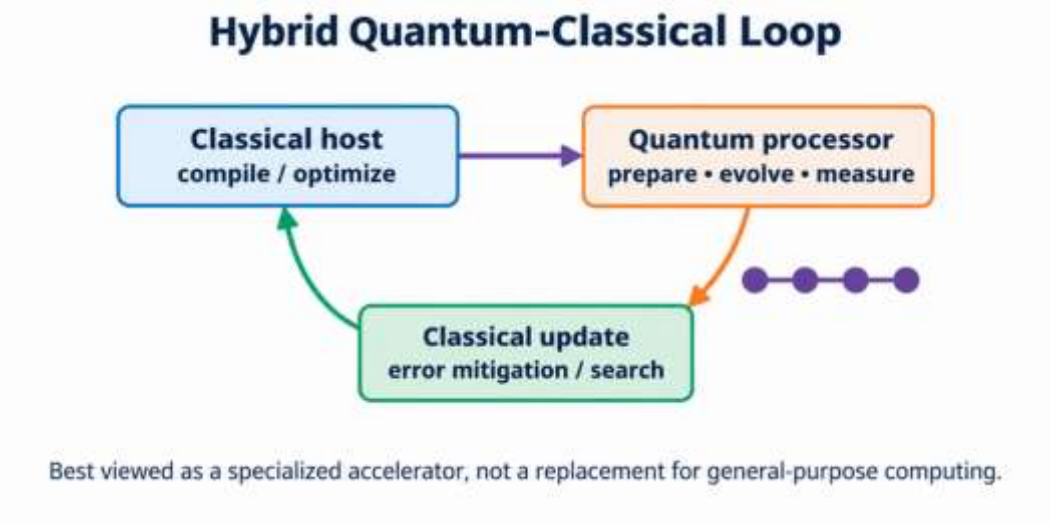


Fig. 8. Hybrid quantum-classical optimization and error-management loop.

Programmable superconducting processors have demonstrated tasks that are difficult to reproduce with specific classical simulation methods [21], while subsequent work has shifted attention from isolated sampling milestones to error correction. Surface-code experiments now show operation below a threshold where increasing code distance can suppress logical errors [22]. Nevertheless, useful fault-tolerant machines require many physical qubits, fast decoding, cryogenic control, high-fidelity gates, and application algorithms whose total resource estimates are economically plausible. Quantum processors should consequently be evaluated like highly specialized accelerators: by end-to-end time, accuracy, energy and infrastructure, including classical control and error-correction overhead.

VIII. APPROXIMATE, STOCHASTIC, AND PROBABILISTIC COMPUTING

Not every application requires every output bit to be exact. Multimedia processing, machine learning, sensing, optimization, and human-facing analytics often tolerate bounded numerical error or naturally represent uncertainty. Approximate computing exploits this tolerance across software, arithmetic units, memories, voltage margins, and communication. The objective is not uncontrolled inaccuracy; it is to spend energy and latency in proportion to the value of precision. A system may reduce mantissa width, skip low-impact operations, use approximate storage, or dynamically select an accuracy level under an application-defined quality contract [28].

Stochastic computing represents a value by the probability of ones in a bit stream. Simple logic gates can then implement multiplication or scaled addition, providing graceful error behavior and potential tolerance to transient faults. Its cost is longer streams, correlation management, random-number generation, and conversion to and from binary representation [29]. Probabilistic bits and annealing-style machines take a related but different route: tunable stochastic elements explore distributions or low-energy configurations, which can match inference and combinatorial optimization but do not provide deterministic acceleration for arbitrary programs.

Approximation is best understood as a cross-cutting method rather than a replacement architecture. Analog CIM, photonics, neuromorphic dynamics, and NISQ quantum hardware all expose noise or limited precision; algorithmic tolerance can convert those nonidealities from defects into design parameters. The danger is silent quality loss. Deployable systems require explicit error budgets, confidence estimates, fallbacks for safety-critical phases, and verification that spans hardware randomness and application semantics.

TABLE II CROSS-PARADIGM COMPARISON

Paradigm	Operand / state representation	Execution trigger	Best-matched workload traits	Principal barriers	Relative maturity
Near-memory	Digital words near DRAM/HBM	Host command / local cores	Low arithmetic intensity; high internal bandwidth	Coherence, address translation, offload granularity	Prototype to product features
Digital CIM	Bits/words in memory arrays	Array command	Bitwise analytics, search, reductions, exact low-precision ops	Array modification, data layout, portability	Research prototypes

Paradigm	Operand / state representation	Execution trigger	Best-matched workload traits	Principal barriers	Relative maturity
Analog CIM	Conductance/charge + voltages	Applied vector	Large reused matrix-vector products	ADC/DAC, variability, drift, endurance, precision	Advanced prototypes
Neuromorphic	Spikes + local neuron/synapse state	Asynchronous event	Sparse temporal sensing, control, always-on learning	Training, mapping, benchmark comparability	Research systems / niche deployment
Dataflow / CGRA	Tokens on graph or spatial routes	Operand readiness / static schedule	Regular pipelines, streaming, tensor and DSP kernels	Compiler complexity, irregular control, remap overhead	Commercial accelerators and prototypes
Photonic	Optical amplitude/phase/wavelength	Modulated light propagation	High-throughput linear transforms and communication	Conversion, nonlinearity, thermal calibration, integration	Rapidly advancing prototypes
Approx. / stochastic	Reduced precision, random bit streams, probability	Quality contract / stochastic evolution	Error-tolerant media, inference, optimization	Error control, conversion, reproducibility, safety	Used selectively in products and research
Quantum	Complex amplitudes of qubits	Gate sequence + measurement	Problems with proven quantum structure	Errors, qubit count, I/O, cryogenics, algorithms	Early experimental systems

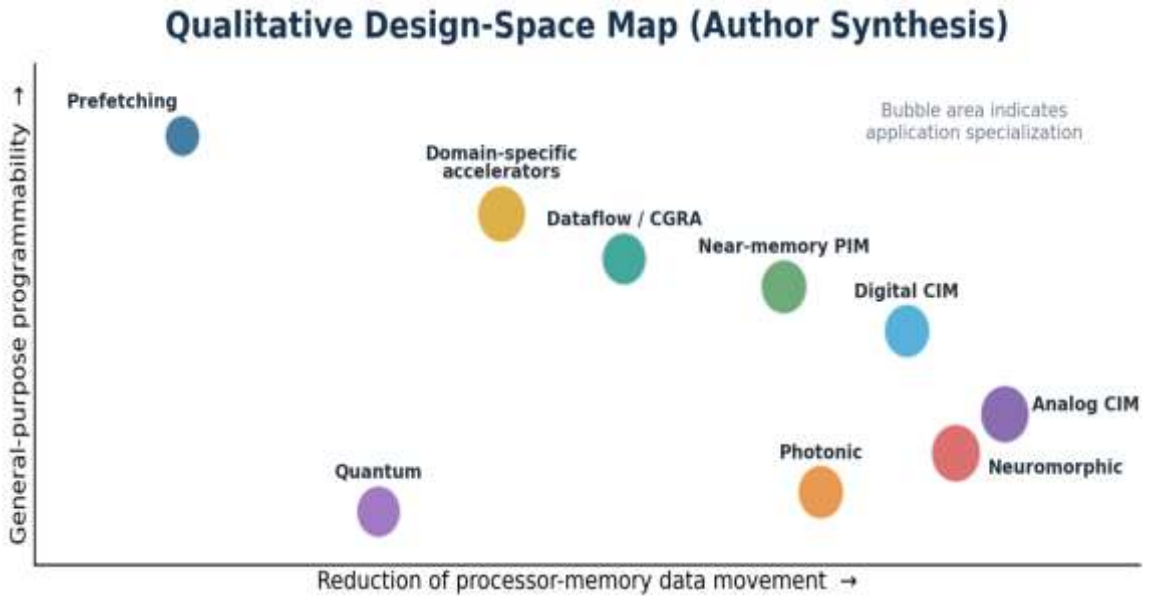


Fig. 9. Qualitative design-space map. Positions are an author synthesis, not benchmark results; bubble size indicates application specialization.

IX. SYNTHESIS: WHAT "BEYOND" ACTUALLY CHANGES

Table II and Fig. 9 expose three recurring strategies. First, memory-centric methods reduce distance between state and arithmetic. Second, dataflow and neuromorphic methods reduce the energy of instruction orchestration by making readiness or events the trigger. Third, photonic and quantum methods

harness a physical process that naturally computes or communicates a desired transformation. These strategies are complementary. A neuromorphic tile may use nonvolatile synapses; a photonic accelerator still needs electronic memory; and a quantum processor depends on a conventional host.

The horizontal axis of Fig. 9 captures reduction of processor- memory data movement, while the vertical axis reflects general-purpose programmability. Conventional multicore CPUs occupy the programmable but movement-intensive corner. Near-memory processing moves right with modest software disruption. Analog CIM and neuromorphic systems can move farther right but trade precision or generality. Photonic and quantum engines are highly specialized: their value is determined by whether a workload exposes the matching mathematical structure. This map also explains why peak operations per second is a poor universal metric. Each substrate counts operations differently and may exclude conversion, communication, training, cooling, or error management.

TABLE III REPRESENTATIVE MILESTONES AND THEIR ARCHITECTURAL LESSON

Work / system	Year	Paradigm	Architectural lesson
Prefetch buffers [5]	1990	Conventional	Prediction can overlap memory latency without changing semantics.
WaveScalar [14]	2003	Dataflow	Operand readiness can replace centralized instruction issue.
TrueNorth [11]	2014	Neuromorphic	Large event-driven fabrics can integrate state and communication.
Coherent nanophotonics [17]	2017	Photonic	Interference can directly realize neural linear transforms.
TPU [16]	2017	Domain-specific	Systolic dataflow and managed memory yield production-scale gains.
Analog-memory training [10]	2018	Analog CIM	Algorithm-device co-design can recover accuracy under nonidealities.
Loihi [12]	2018	Neuromorphic	Programmable local learning enables adaptive event processing.
Quantum sampling [21]	2019	Quantum	Programmable quantum hardware can enter classically difficult regimes.
Below-threshold QEC [22]	2025	Quantum	Logical-error suppression, not raw qubit count, is the key scaling milestone.

X. OPEN CHALLENGES

A. End-to-End Accounting and Benchmarking

A credible evaluation must include the host, memory movement, data-format conversion, compilation, calibration, cooling where applicable, and quality of result. Array-level tera-operations per second can be misleading when converters dominate; synaptic-operation energy can omit spike encoding; and quantum gate time can omit sampling and decoding. Benchmarks should report latency distributions, throughput, total energy, accuracy, reliability, area or packaging cost, and the fraction of the application that remains on the host. Common baselines and open workloads are essential to distinguish architectural advantage from favorable accounting.

B. Programming Models, Compilers, and Runtime Scheduling

The central software question is how much hardware specificity must leak into programs. Memory-centric systems need data-placement and consistency semantics. CGRAs require placement and routing. Neuromorphic systems need temporal encodings and learning rules. Photonic compilers must account for precision and calibration, while quantum compilers manage topology, noise, and error-correction resources. A productive stack should express intent at a stable level, specialize aggressively below it, and preserve a fallback path on conventional hardware. Runtime systems must decide whether the cost of conversion and transfer is justified for the current input size and hardware state.

C. Reliability, Security, and Isolation

Nontraditional substrates introduce nontraditional faults. Analog weights drift; optical components vary with temperature; nonvolatile memories wear; asynchronous events are sensitive to timing assumptions; and quantum states decohere. Fault tolerance must therefore be co-designed with algorithms rather than

added as an afterthought. Security is equally important. Shared accelerators create timing and contention channels, memory-centric execution complicates protection boundaries, and analog or approximate behavior can expose fault-injection opportunities. Isolation, attestation, secure compilation, and data erasure must extend across chiplets and heterogeneous memories.

D. Manufacturability and Sustainable Scaling

A prototype becomes a platform only when yield, packaging, test, repair, tool support, and supply-chain cost are manageable. Three-dimensional integration improves proximity but increases thermal density. Photonics requires reliable electronic integration and packaging. Cryogenic quantum control has substantial infrastructure requirements. Sustainability should be evaluated over the complete lifecycle: embodied energy, cooling, utilization, and the possibility that specialized hardware becomes obsolete before its manufacturing cost is amortized.

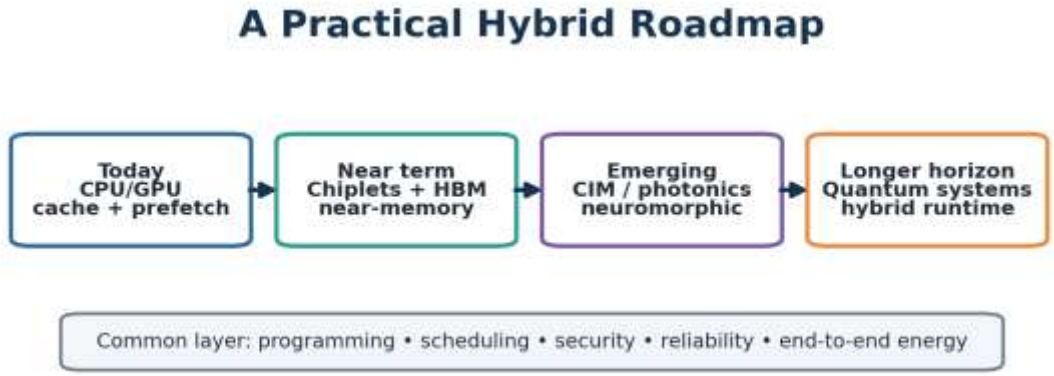


Fig. 10. A practical roadmap toward heterogeneous beyond-von Neumann systems.

TABLE IV PRIORITIES FOR A DEPLOYABLE HETEROGENEOUS ECOSYSTEM

Research layer	Near-term priority	Why it matters across paradigms
Interfaces	Coherent memory, standard command queues, discoverable capabilities	Allows accelerators to be composed rather than programmed as isolated appliances.
Compilers	Cost-aware partitioning, precision selection, layout and route generation	Prevents transfer or conversion overhead from erasing hardware gains.
Runtime	Dynamic placement using load, temperature, accuracy and device health	Matches each phase to the substrate currently offering the best system-level value.
Verification	Cross-domain numerical contracts and fault models	Makes approximate, analog, event-driven and quantum results auditable.
Security	Isolation, attestation, controlled sharing and side-channel accounting	Protects data when computation moves into memories and shared specialized fabrics.
Metrics	Application-level energy, latency, quality, cost and lifecycle reporting	Enables fair comparison and discourages boundary-based efficiency claims.

XI. A PRACTICAL HYBRID ROADMAP

Figure 10 summarizes the likely transition. In the near term, multicore CPUs and GPUs remain the control plane, while chiplets, high-bandwidth memory, and domain-specific accelerators handle dominant kernels. Near-memory engines and CIM arrays will be adopted where data reuse and precision requirements justify them. Neuromorphic and photonic components will grow first in niches that naturally match sparse events or linear transforms. Fault-tolerant quantum processors, if realized economically, will remain attached accelerators for selected algorithms rather than replacements for operating systems, databases, and general-purpose code.

The unifying architectural requirement is a common orchestration layer. A future application may preprocess data on a CPU, execute dense layers on a digital or photonic matrix engine, run sparse temporal

inference on a neuromorphic tile, and invoke a quantum routine for a small optimization subproblem. The system must move data deliberately, preserve security domains, negotiate numerical precision, and decide when acceleration is not worthwhile. In this sense, the era beyond von Neumann is less about a single successor and more about making heterogeneity a first-class, programmable property.

XII. SELECTING A SUBSTRATE: WORKLOAD WALKTHROUGHS

A useful architecture review should end with a selection process rather than a ranking. Consider first a dense neural-network layer. When matrices are modest, frequently changing, or require high numerical precision, a CPU or GPU with effective tiling, caching, and prefetching may provide the best total cost. A digital systolic or spatial accelerator becomes attractive when operators and tensor shapes are stable enough to reuse a specialized data path. Analog CIM becomes a candidate when weights can remain resident, matrix-vector products dominate, and application accuracy tolerates calibrated low-precision accumulation. Photonics moves to the front only when a sufficiently large or repeated linear transform amortizes modulation, detection, data conversion, and thermal control. The same mathematical kernel can therefore select different hardware at different batch sizes and accuracy targets.

Now consider graph traversal or sparse analytics. Addresses depend on data, locality changes with the frontier, and cache lines may contain few useful elements. A conventional prefetcher often has low coverage or generates harmful traffic. Near-memory processing can instead scan adjacency structures and perform filtering or reduction beside high-bandwidth memory, returning only compact results. A CGRA may help after a regular sparse format and pipeline are established, but dynamic load imbalance and unpredictable branches can waste mapped resources. When the graph represents a stream of temporal events rather than a bulk data structure, a neuromorphic engine may be appropriate because inactivity is naturally cheap and state is updated only on arrivals.

Optimization provides a third cautionary example. Approximate arithmetic can accelerate objective evaluation when error bounds are understood; stochastic or annealing hardware can explore candidate distributions; and quantum routines may accelerate carefully structured subproblems. Yet the decisive metric is not the speed of the inner primitive. It is time-to-quality after encoding, transfers, repeated samples, verification, and any classical refinement. A small, well-vectorized classical solver can outperform an exotic substrate when the problem instance is too small, the mapping is dense, or solution checking dominates.

These walkthroughs suggest a general rule: select a beyond-von Neumann engine only after identifying the dominant data movement, the stable mathematical primitive, the required quality of result, and the boundary costs. The preferred design may also change during execution. A runtime could use a CPU for control-heavy phases, a near-memory engine for filtering, a matrix accelerator for dense kernels, and a conventional fallback when device health or calibration degrades. Such phase-aware composition is more realistic than assigning an entire application to one paradigm.

TABLE V PRACTICAL WORKLOAD-TO-SUBSTRATE SCREENING

Observed workload property	First substrate to evaluate	Reject or re-evaluate when
Working set fits and addresses are predictable	Multicore CPU/GPU with caches and prefetching	Bandwidth or data movement still dominates after tuning
Low arithmetic intensity over large resident data	Near-memory processing	Offload, coherence, or placement overhead exceeds filtered result size
Repeated low-precision matrix-vector products	Analog CIM	Converters, drift, updates, or accuracy recovery dominate
Exact bitwise/reduction operations inside arrays	Digital CIM	Memory modifications or layout constraints reduce usable capacity
Sparse temporal events with persistent local state	Neuromorphic	Encoding becomes dense or training requires frequent host intervention
Regular producer-consumer pipelines	Dataflow / CGRA / systolic engine	Branches, dynamic dependence, or remapping destroy utilization
Large reused linear transforms or bandwidth-limited links	Photonic accelerator/interconnect	Electronic conversion, tuning, or nonlinearity dominates
Error-tolerant media, inference, or search	Approximate / stochastic hardware	Quality cannot be bounded, audited, or safely recovered

Observed workload property	First substrate to evaluate	Reject or re-evaluate when
Problem has a credible quantum algorithm and resource estimate	Quantum accelerator	State preparation, error correction, sampling, or verification dominates

XIII. CONCLUSION

Caches and prefetchers have prolonged the effectiveness of the conventional multicore model by exploiting and predicting locality. They remain essential because most software will continue to execute on stored-program processors. Their limitation is that they optimize the movement of data between physically separate state and computation. Beyond-von Neumann methods attack the problem at deeper levels: computing where data resides, triggering work by events or operand readiness, spatially mapping communication, or using optical and quantum physics to realize selected operations.

No surveyed paradigm simultaneously maximizes programmability, precision, maturity, and data-movement reduction. Consequently, the most credible future is heterogeneous and layered. Success will depend as much on interfaces, compilers, reliability, security, and honest end-to-end metrics as on device-level efficiency. The defining achievement of the next computing era will not be the disappearance of the von Neumann machine, but its transformation into an orchestrator of diverse computational substrates.

References

[1] J. von Neumann, First Draft of a Report on the EDVAC, Moore School of Electrical Engineering, Univ. of Pennsylvania, 1945.

[2] W. A. Wulf and S. A. McKee, 'Hitting the memory wall: Implications of the obvious,' ACM SIGARCH Computer Architecture News, vol. 23, no. 1, pp. 20-24, 1995, doi: 10.1145/216585.216588.

[3] M. D. Hill and M. R. Marty, 'Amdahl's law in the multicore era,' Computer, vol. 41, no. 7, pp. 33-38, 2008, doi: 10.1109/MC.2008.209.

[4] J. L. Hennessy and D. A. Patterson, 'A new golden age for computer architecture,' Commun. ACM, vol. 62, no. 2, pp. 48-60, 2019, doi: 10.1145/3282307.

[5] N. P. Jouppi, 'Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers,' in Proc. 17th Annu. Int. Symp. Computer Architecture (ISCA), 1990, pp. 364-373, doi: 10.1145/325096.325162.

[6] M. Bakhshalipour, S. Tabaeiaghdaei, P. Lotfi-Kamran, and H. Sarbazi-Azad, 'Evaluation of hardware data prefetchers on server processors,' ACM Comput. Surv., vol. 52, no. 3, Art. no. 52, 2019, doi: 10.1145/3312740.

[7] W. Sun et al., 'A survey of computing-in-memory processor: From circuit to application,' IEEE Open J. Solid-State Circuits Soc., vol. 4, pp. 25-42, 2024, doi: 10.1109/OJSSCS.2023.3328290.

[8] M. A. Asifuzzaman, F. R. Taher, and M. R. R. Khan, 'A survey on processing-in-memory techniques: Advances and challenges,' Integration, vol. 91, pp. 31-55, 2023.

[9] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, 'Memory devices and applications for in-memory computing,' Nat. Nanotechnol., vol. 15, pp. 529-544, 2020, doi: 10.1038/s41565-020-0655-z.

[10] S. Ambrogio et al., 'Equivalent-accuracy accelerated neural-network training using analogue memory,' Nature, vol. 558, pp. 60-67, 2018, doi: 10.1038/s41586-018-0180-5.

[11] P. A. Merolla et al., 'A million spiking-neuron integrated circuit with a scalable communication network and interface,' Science, vol. 345, no. 6197, pp. 668-673, 2014, doi: 10.1126/science.1254642.

[12] M. Davies et al., 'Loihi: A neuromorphic manycore processor with on-chip learning,' IEEE Micro, vol. 38, no. 1, pp. 82-99, 2018.

[13] C. D. Schuman et al., 'Opportunities for neuromorphic computing algorithms and applications,' Nat. Comput. Sci., vol. 2, pp. 10-19, 2022, doi: 10.1038/s43588-021-00184-y.

[14] S. Swanson et al., 'WaveScalar,' in Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO-36), 2003, pp. 291-302.

[15] A. Podobas, K. Sano, and S. Matsuoka, 'A survey on coarse-grained reconfigurable architectures from a performance perspective,' IEEE Access, vol. 8, pp. 146719-146743, 2020.

[16] N. P. Jouppi et al., 'In-datacenter performance analysis of a tensor processing unit,' in Proc. 44th Annu. Int. Symp. Computer Architecture (ISCA), 2017, pp. 1-12, doi: 10.1145/3079856.3080246.

[17] Y. Shen et al., 'Deep learning with coherent nanophotonic circuits,' Nat. Photonics, vol. 11, pp. 441-446, 2017, doi: 10.1038/nphoton.2017.93.

[18] T. Fu et al., 'Optical neural networks: Progress and challenges,' Light Sci. Appl., vol. 13, Art. no. 263, 2024, doi: 10.1038/s41377-024-01590-3.

[19] H. Zhang et al., 'Integrated platforms and techniques for photonic neural networks,' npj Nanophotonics, vol. 2, Art. no. 40, 2025, doi: 10.1038/s44310-025-00088-z.

[20] J. Preskill, 'Quantum computing in the NISQ era and beyond,' Quantum, vol. 2, Art. no. 79, 2018, doi: 10.22331/q-2018-08-06-79.

[21] F. Arute et al., 'Quantum supremacy using a programmable superconducting processor,' Nature, vol. 574, pp. 505-510, 2019, doi: 10.1038/s41586-019-1666-5.
[22] Google Quantum AI and Collaborators, 'Quantum error correction below the surface code threshold,' Nature, vol. 638, pp. 920-926, 2025, doi: 10.1038/s41586-024-08449-y.
[23] D. Kimovski et al., 'Beyond von Neumann in the computing continuum: Architectures, applications, and future directions,' IEEE Internet Comput., vol. 28, no. 3, pp. 6-16, 2024, doi: 10.1109/MIC.2023.3301010.
[24] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, 'Efficient processing of deep neural networks: A tutorial and survey,' Proc. IEEE, vol. 105, no. 12, pp. 2295-2329, 2017, doi: 10.1109/JPROC.2017.2761740.
[25] M. Le Gallo et al., 'Mixed-precision in-memory computing,' Nat. Electron., vol. 1, pp. 246-253, 2018.
[26] W. Haensch, T. Gokmen, and R. Puri, 'The next generation of deep learning hardware: Analog computing,' Proc. IEEE, vol. 107, no. 1, pp. 108-122, 2019.
[27] R. Kudithipudi et al., 'Neuromorphic computing at scale,' Nature, 2025, doi: 10.1038/s41586-024-08253-8.
[28] S. Mittal, 'A survey of techniques for approximate computing,' ACM Comput. Surv., vol. 48, no. 4, Art. no. 62, 2016, doi: 10.1145/2893356.
[29] A. Alaghi and J. P. Hayes, 'Survey of stochastic computing,' ACM Trans. Embedded Comput. Syst., vol. 12, no. 2s, Art. no. 92, 2013, doi: 10.1145/2465787.2465794.