Cross-Layer Design Approaches of Stateful Digital Memristive In-Memory Computing

Fabian Seiler[†], Hongrong Hu[+], Ryan Wong[*], Kerem Kagan Güclü[+], Yiqiu Sun[*],
Saugata Ghose[*], Jasmin Aghassi-Hagmann[+], and Nima TaheriNejad[†]
[†]Heidelberg University, [+]Karlsruhe Institute of Technology (KIT), [*]University of Illinois Urbana-Champaign
corresponding author: fabian.seiler@ziti.uni-heidelberg.de

***Abstract*—Digital memristive In-Memory Computing (IMC) reduces data movement by executing Boolean and arithmetic operations directly within resistive memory arrays through Stateful Logic (SL), where devices act as both storage and computational elements. This enables highly parallel general-purpose computation, but also breaks conventional abstraction boundaries: device non-idealities affect logic execution and scalability, circuit-level paradigms shape architectural trade-offs, and workload requirements constrain lower-layer design choices. This paper argues that future progress will depend on cross-layer co-design rather than isolated optimization of devices, circuits, or architectures. We provide a cross-layer perspective on the digital memristive computing stack, highlight key dependencies between abstraction layers, and outline directions toward scalable, reliable, and programmable digital memristive IMC systems.**

I. Introduction

Modern data-centric workloads are increasingly constrained by the energy and latency costs of moving data between memory and processing units [1]. To address this bottleneck, In-Memory Computing (IMC), a.k.a. Processing-Using-Memory (PUM), seeks to perform computation directly where data resides, reducing data movement and improving energy efficiency. Among emerging IMC technologies, Resistive Random Access Memories (RRAMs) are particularly attractive due to their non-volatility, high integration density, CMOS compatibility, and ability to support massively parallel operations within crossbar arrays [2]–[5]. While most memristive IMC systems target analog matrix–vector multiplication for neural-network acceleration [6], digital memristive IMC enables general-purpose computation through Stateful Logic (SL), where memristive devices simultaneously serve as storage and computational elements. By performing Boolean and arithmetic operations directly within memory arrays, SL tightly integrates memory and computation while minimizing data movement [4], [5], [7], [8].

However, this tight integration also breaks conventional abstraction boundaries. Unlike CMOS systems, where devices, circuits, and architectures can often be optimized largely independently, digital memristive computing exhibits strong cross-layer dependencies. Device characteristics such as resistance ranges, switching thresholds, and variability directly influence SL execution and reliability [9]–[11]. Circuit-level computing paradigms determine operation complexity, communication patterns, and synchronization requirements, which subsequently shape datapath microarchitectures and architectural organizations [12]–[16]. Conversely, workload requirements such as accuracy, throughput, latency, and error tolerance propagate downward and constrain architectural, circuit-level, and even device-level design choices. For example, device variability affects logic correctness and scheduling, arithmetic complexity influences architectural organization, and application-level error tolerance can enable approximate or stochastic computing paradigms.

Despite these strong interdependencies, research is often conducted within individual abstraction layers, with device, circuit, and architectural efforts largely optimized in isolation. As a result, opportunities and constraints that emerge at the interfaces between layers are frequently overlooked. This paper argues that future progress in digital memristive IMC will increasingly depend on cross-layer co-design. To support researchers working at different levels of the stack, we provide a structured overview of major RRAM technologies (Section II), circuit-level computing paradigms (Section III), and architectural organizations for scalable digital IMC (Section IV). We then discuss the dependencies between these layers and outline emerging research directions toward scalable, reliable, and programmable digital memristive computing systems (Section V).

II. RRAM Devices and Reliability Challenges

A. Overview of Memristive Device Technologies

The most widely investigated resistive memory technologies for digital IMC include Valence Change Mechanism (VCM) [17], Electrochemical Metallization (ECM) [18], and Self-Directed Channel (SDC) [19] devices. Although all rely on electrically induced resistance changes, they differ significantly in their filament formation mechanisms, switching thresholds, dynamic range (R_{off}/R_{on}), endurance, and variability characteristics. These properties directly influence device-level metrics such as switching energy, latency, reliability, and operating margins.

VCM devices rely on oxygen-vacancy migration in metal oxides such as HfO_x and TaO_x, forming conductive filaments between electrodes through local valence changes induced by redox reactions and ion migration [17] (Figure 1a). Applying voltages of opposite polarities switches the device between its Low Resistance State (LRS) and High Resistance State (HRS)

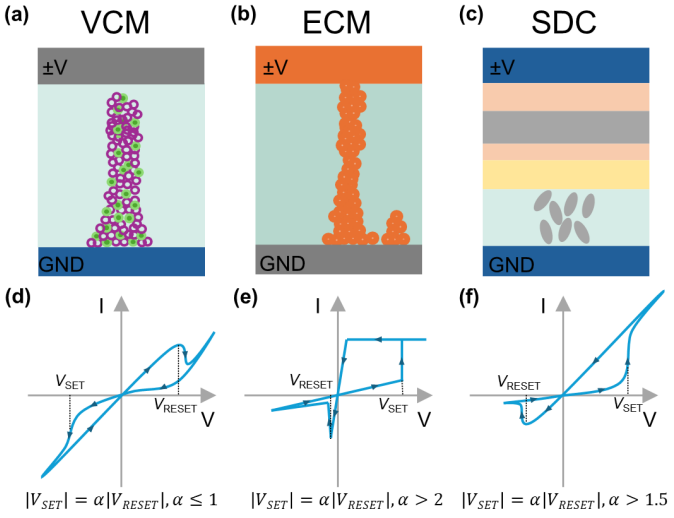

Fig. 1: Overview of common bipolar RRAM types: (a)-(c) show the resistive switching mechanism of VCM, ECM, and SDC, while (d)-(f) illustrate representative I-V characteristics of the three types.

(SET and RESET), as shown in Figure 1d. Because switching is governed by gap modulation between the filament and electrode, VCM devices exhibit relatively symmetric switching behavior with $|V_{SET}| = \alpha|V_{RESET}|$, $\alpha \leq 1$ [17]. They further provide strong CMOS compatibility, high endurance, and mature fabrication processes [2], making them attractive for large-scale SL arrays. However, many VCM technologies require electroforming and can exhibit cycle-to-cycle variability arising from stochastic ion migration, which directly impacts logic reliability and operating margins [3].

ECM devices also rely on filament formation and dissolution (Figure 1b), but through electrochemical deposition of active metals such as Ag or Cu [11], [20]. The fast migration rate of metallic cations enables low switching voltages and currents, making ECM devices attractive for energy-efficient SL. Formation and rupture of highly conductive metallic filaments produce large resistance ranges, often reaching $R_{off}/R_{on} > 10^6$. ECM devices further exhibit pronounced switching asymmetry with $|V_{SET}| = \alpha|V_{RESET}|$, $\alpha > 2$, as shown in Figure 1e, because SET and RESET are governed to different extents by redox reactions (and ion migration) and Joule heating. While these characteristics can benefit certain SL families, stochastic filament growth introduces significant temporal and spatial variability that can reduce logic reliability and timing margins.

SDC devices employ more complex multilayer structures (Figure 1c) in which Ag ions can enter the active layer and form conductive channels through cluster agglomeration rather than metallic filament growth [19]. Switching is achieved by enriching or depleting Ag within these channels through applying proper voltages, resulting in more gradual resistance modulation than in ECM devices. Consequently, SDC devices exhibit a moderate dynamic range, less pronounced threshold asymmetry with $|V_{SET}| = \alpha|V_{RESET}|$, $\alpha > 1.5$, and more analog switching behavior, as illustrated in Figure 1f. These characteristics can potentially offer improved endurance, cycling reliability, and lower initialization overheads. Nevertheless, SDC technologies remain less mature compared to VCM and ECM approaches, with fewer large-scale integration demonstrations and device models currently available.

B. Implications for Stateful Logic

With SL, the non-volatile resistive states of the RRAM device are used to represent the information of inputs and the output. It additionally enables voltage-divider circuits that execute the Boolean operations via conditional state changes. Because the intrinsic device properties and SL circuit design rules are strongly coupled, a cross-layer co-design methodology between the device and circuit levels can facilitate rational and practical implementation [7], [10], [21], [22]. Therefore, the underlying device mechanism directly determines the circuit performance, including latency, energy consumption, variability, and computational reliability.

The exemplary FELIX-OR [5] circuit (Figure 2a) conditionally switches the output device from HRS to LRS when any input is in LRS (Figure 2b). Reliable execution requires stable input states during operation, favoring devices with switching characteristics of approximately $|V_{SET}/V_{RESET}| < 2$ [5]. Consequently, VCM devices naturally provide favorable operating conditions, whereas the stronger switching asymmetry of ECM devices increases the likelihood of unintended input-state transitions and logic failures [10], [21]. Interestingly, the opposite trend can occur for MAGIC-NOR [7], which requires stronger threshold asymmetry and therefore favors devices with $|V_{SET}/V_{RESET}| > 2$ [21], [22]. These examples illustrate a key cross-layer observation: no memristor technology is universally optimal for SL; instead, device characteristics and logic implementations must be considered jointly. Consequently, device non-idealities directly propagate to logic execution, where they accumulate over long operation sequences and affect reliability, latency, and throughput [3], [9]–[11]. Importantly, how device-level non-idealities are handled depends strongly on the selected computational paradigm discussed in the following section.

III. Circuit-Level Computing Paradigms

A. Exact Stateful Logic

Memristive logic approaches can be classified according to whether information is represented as voltages or resistive states. SL stores both inputs and outputs directly as memristive states, allowing computation entirely within the memory array and eliminating the need for conversion between stored states and voltage signals, thus reducing peripheral complexity, data movement, and energy consumption. Several SL families have been proposed, such as IMPLY, MAGIC, FELIX, and OSCAR [4], [5], [7], [22]. Despite different circuit structures, all realize Boolean functionality through conditional switching of memristive devices, where voltage-divider effects determine whether an output device changes state (Figure 2a-b).

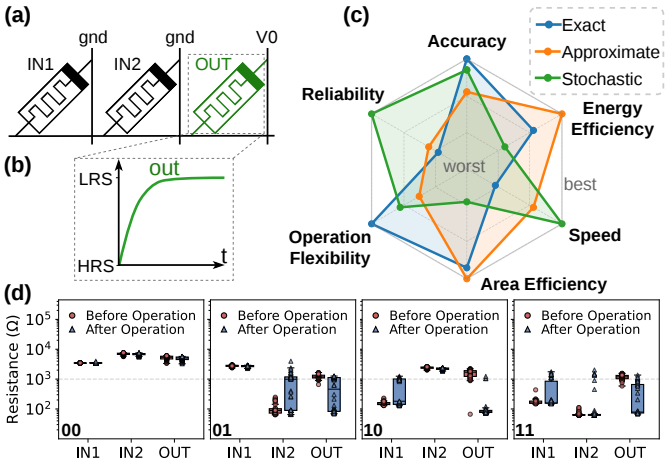

Fig. 2: SL computing paradigms: (a) exemplary logic gate within the crossbar [5], (b) conditional switching of output iff $IN_1 \vee IN_2$, (c) comparison of SL paradigms, (d) measurement of FELIX-OR gate using real ECM memristors that were inkjet-printed and featured the same device structure as in [20].

Correct operation requires carefully selecting pulse amplitudes, pulse durations, and peripheral components (e.g., gate resistances [4]). These operating conditions are strongly influenced by the underlying device characteristics, such as switching thresholds, resistance range, and variability, making SL implementations inherently device-dependent [4], [5], [7], [21]. Device-level effects such as cycle-to-cycle and device-to-device variability, resistance drift, and stochastic switching reduce the available noise margins and can lead to missing/unwanted output transitions or unintended switching of input devices (Figure 2d) [20].

These limitations become particularly pronounced for arithmetic functions. Since SL natively supports only a small set of Boolean operations known as Micro-Operations (µops), addition, multiplication, and other arithmetic kernels must be decomposed into long sequences of dependent µops [4], [5], [7], [16], [23]. As µop depth increases, latency accumulates, and device-level non-idealities increasingly affect system-level reliability and scalability. Consequently, exact SL provides deterministic IMC, but exposes a fundamental trade-off between correctness, efficiency, and scalability. Most SL µops require the initialization of a dedicated output memristor (e.g., HRS for FELIX-OR) that introduces additional overhead, particularly for complex arithmetic workloads, impacting the design choices of datapath microarchitectures. Exact SL establishes the baseline for digital memristive computing by providing exact Boolean functionality directly within memory arrays. However, device-level non-idealities constrain its operating margins, while the sequential nature of stateful arithmetic limits its throughput and scalability. These limitations have motivated the combination of SL with other emerging paradigms that address different aspects of the problem: Approximate Computing (AxC) trades accuracy for improved efficiency, while Stochastic Computing (SC) leverages probabilistic representations to achieve highly parallel, low-latency, and variability-tolerant computation.

B. Approximate Stateful Logic

Many data-intensive workloads can tolerate bounded computational errors without significantly affecting output quality. AxC exploits this property to improve performance and energy efficiency by selectively relaxing correctness requirements [12], [13]. For SL, AxC provides a natural response to one of the key limitations of exact execution: complex arithmetic functions require long operation chains of dependent Boolean operations. Replacing exact arithmetic units with simplified counterparts (e.g., OR to replace full adders [13]) significantly reduces operation count, area, and energy, while maintaining acceptable application-level accuracy. Importantly, these approximations are introduced intentionally at the arithmetic level and therefore differ fundamentally from errors caused by device non-idealities. However, AxC reduces the cumulative probability of switching-related failures, since it needs fewer operations. The benefits of approximation are inherently application dependent and are most pronounced for arithmetic kernels in error-tolerant workloads [12]. Moreover, the final trade-off depends not only on arithmetic error but also on how an approximate function maps to specific SL families and datapath microarchitectures. Consequently, approximate SL represents a direct cross-layer interaction between workload requirements, circuit design, and architectural execution costs.

C. Stochastic Computing Using Stateful Logic

SC replaces conventional binary representations with probabilistic bitstreams, where a value is encoded by the probability of observing a logical '1' (e.g., 1101 represents 3/4) [14]. This representation can greatly simplify selected arithmetic operations, e.g., the multiplication of two uncorrelated bitstreams can be implemented in constant logic depth using a single AND gate per bit. Consequently, SC trades arithmetic complexity for representation complexity (additional storage, communication bandwidth, synchronization, and correlation management). SC is particularly attractive for RRAM because information is distributed across many equally weighted bits, making computations naturally tolerant to individual switching errors. Furthermore, the inherent stochasticity can be exploited directly for bitstream generation [14]. Consequently, SC shifts the dominant challenges from device reliability toward efficient bitstream generation and management at the circuit and architecture levels.

Overall, exact SL, approximate SL, and SL-based SC represent three complementary responses to the limitations of digital memristive computing. Exact SL prioritizes exact computation but suffers from tight operating margins and long arithmetic sequences. Approximate SL reduces arithmetic complexity for error-tolerant workloads, while SC exploits probabilistic representations to simplify selected operations and improve robustness. The resulting trade-offs are summarized in Figure 2c.

IV. Architectural Design Choices

A. Organizing Arrays Into Datapaths

While circuit-level paradigms define how SL μops are executed, datapath microarchitectures determine how these μops are organized across crossbar arrays to expose parallelism, improve throughput, and manage communication overheads. Datapath organizations can be broadly categorized into three types, as illustrated in Figure 3 [15], [16]. Bit-serial datapaths apply a single μop across an entire crossbar, selecting one or more columns from the crossbar and performing the same μop to be executed concurrently across all rows of the selected columns [15]. Though simple, bit-serial's scalability is ultimately limited by sequential execution and the maximum current that can be safely supplied to arrays while avoiding electromigration effects [16]. Bit-parallel datapaths partition a crossbar into multiple independently controlled regions that execute μops concurrently [15]. This increases throughput and reduces latency, but requires additional control infrastructure (e.g., for per-partition opcodes), partition management, and communication mechanisms between partitions. Bit-pipelined datapaths distribute computations across per-bit-position stages connected by dedicated buffers, enabling efficient execution of arithmetic workloads with localized dependencies (e.g., ripple-carry adders) [16], [22]. This improves throughput and reduces communication overhead while reusing control infrastructure, but requires pipeline-aware SL algorithms and more sophisticated scheduling and synchronization mechanisms.

These organizations primarily differ in how they trade off concurrency against communication and control overhead. Exact SL, with its long dependency-rich execution sequences, often benefits from parallel and pipelined organizations that mitigate computational bottlenecks, whereas approximate and stochastic paradigms can reduce arithmetic complexity, altering both execution and communication requirements. Consequently, datapath microarchitectures directly shape higher-level concerns such as data movement, scheduling, control complexity, and programmability.

B. Coordinating Operations Across Datapaths

1) Architectural Organization and Communication: While SL μops are executed locally within the datapath organizations in Section IV-A, realistic workloads require larger storage capacities, greater parallelism, and more flexible execution. Early architectures provided a flat hierarchy of memory banking, where each bank contained a scaled-up array [24]. To avoid electromigration and device constraints, more recent works utilize multi-level hierarchies that efficiently use smaller arrays [16], [22]. This shift allows workloads to scale beyond the capacity of individual arrays while hiding many low-level architectural constraints from software.

A popular approach for exact SL architectures is to expose a vector-style instruction interface to software systems, where a single instruction operates on many data elements at once and hides the underlying array organization [16], [22]–[24].

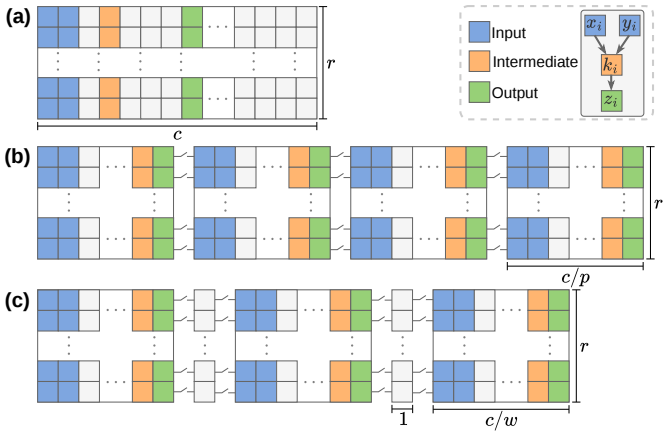

Fig. 3: Datapath organizations for SL: (a) bit-serial, with a $c \times r$ array; (b) bit-parallel, with p partitions; and (c) bit-pipelined, with w stages for a w-bit word connected through $1 \times r$ buffers.

Alternative paradigms such as AxC and SC relax some of these assumptions by reducing operation counts or replacing binary computation with massively parallel bitstream processing [12], [14].

As computations become distributed across multiple arrays, communication increasingly becomes a first-class architectural concern. Pipeline-oriented systems reduce communication overhead through localized data movement and shared control resources, whereas processing-unit organizations provide greater flexibility through shared resources and more general communication mechanisms [24]. The resulting trade-off is between throughput and communication efficiency on one hand, and programmability and flexibility on the other. Consequently, architectural organization, communication interfaces, and circuit-level computing paradigms must be considered jointly when designing scalable digital memristive systems.

2) Programmability and Control: At a high level, an SL-capable architecture exposes millions to billions of ways of parallelism. Manually writing individual SL μops at such an extreme level is impractical for all but the most simple kernels, akin to writing billions of lines of assembly. This has led to the rise of two types of tractable programming models for SL: (1) leveraging Electronic Design Automation (EDA) tools to synthesize the logic for a chain of circuits on the memory arrays [25], or (2) adapting higher-level programming languages that can be compiled down into SL μops [23], [26].

A key challenge with software toolchains and programming models for SL is balancing programmability and efficiency, which can be achieved through flexible abstraction layers. Recent efforts eliminate the need for programmers to manage individual memory arrays, by providing cross-architectural abstractions through which programs express their inherent data-level parallelism in a microarchitecture-agnostic manner, while runtime software and control path hardware dynamically map the parallelism to SL μops [23]. At the hardware–software interface, architectures often expose an Instruction Set Archi-

tecture (ISA) consisting of higher-level instructions (similar to those used for modern CPUs and GPUs), instead of directly encoding logic-family-specific µops, to reduce program binary size and improve portability [16], [23].

Control infrastructure has become a central design concern that ties the software side of architectural abstractions to the hardware side, crossing architecture-, circuit-, and device-level layers. A hardware system with SL capabilities needs to manage the execution of entire applications. To do so, the system must coordinate operations across SL-capable execution units (consisting of one or more datapaths) and other non-SL computing elements (e.g., host CPUs, GPUs, other accelerators). Early accelerator-style approaches rely heavily on a host CPU to manage execution and coordinate computational arrays [24]. More memory-centric systems embed advanced control path hardware directly within the execution units, enabling more autonomous execution, communication, and workload management [22], [23]. At the datapath level, the control path must first convert higher-level program instructions from a program binary into datapath-specific µop sequences [23], and then convert this into sequences of voltage-driven operations that are specific to an underlying device technology [15], [16], [22], [24].

V. Cross-Layer Co-Design and Outlook

A. Cross-Layer Challenges

The preceding sections demonstrate that digital memristive IMC cannot be optimized one abstraction layer at a time. Unlike conventional mature CMOS systems, where abstraction layers can often be developed largely independently, emerging SL directly exposes device behavior to computation. Consequently, design decisions propagate throughout the stack, from materials and devices to circuits, architectures, software, and workloads. Importantly, improvements at one abstraction layer do not necessarily translate into system-level gains. A logic family that minimizes operation count may increase communication overhead, while a device optimized for memory performance may provide unfavorable switching characteristics for computation. Similarly, architectures that maximize throughput may expose programmability and synchronization challenges at the software level. Understanding and managing these trade-offs requires coordinated optimization across multiple abstraction layers rather than isolated improvements within individual layers.

Figure 4 summarizes these bidirectional dependencies. Material compositions determine the switching mechanisms and characteristics of the devices, such as resistance ranges, switching thresholds, endurance, retention, and variability. At the device–circuit interface, these properties directly affect operating margins, logic correctness, latency, and energy consumption of SL operations. As shown by the FELIX and MAGIC examples, device properties can even determine whether practical implementations are feasible. This strong dependency motivates device-aware circuit design supported by realistic device models [11], enabling automated optimization and validation under non-ideal device behavior. We posit that

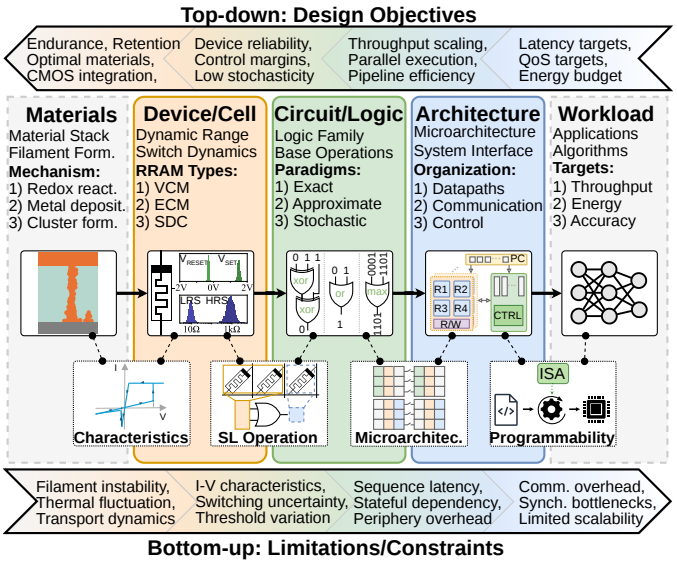

Fig. 4: Cross-layer dependencies in digital memristive IMC systems. Objectives and constraints propagate across abstraction layers, requiring coordinated optimization from materials to workloads.

SL operations should be optimized not only for performance, but also for robustness against device variability and switching uncertainty.

At the circuit–architecture interface, different computing paradigms impose fundamentally different execution requirements. Exact SL often requires long dependency-rich operation sequences that motivate parallel and pipelined organizations to improve throughput and communication efficiency. Approximate SL reduces arithmetic complexity through controlled inaccuracies, whereas SC replaces binary operation chains with massively parallel bitstream processing. Consequently, the optimal microarchitecture depends strongly on the selected computing paradigm, which explicitly accounts for operation sequences, communication patterns, reliability requirements, and workload characteristics [12]–[14]. Automated circuit-level evaluation and SPICE-based validation using realistic models can accelerate this process by exposing how designs translate into latency, energy, robustness, and architectural constraints [9].

At the architecture–workload interface, microarchitectural choices directly influence architectural organization, communication infrastructure, and programmability. As stateful computations are distributed across multiple arrays, communication, synchronization, and control increasingly become dominant contributors to system performance. Recent architectural studies already indicate that these overheads can limit scalability despite being highly efficient in-memory datapaths [23]. Future architectures will therefore require tighter integration between hardware organization, ISA design, compiler support, and application mapping. Depending on workload characteristics, memristive processing units may be deployed either as specialized accelerators [24] or as more tightly integrated computational substrates [16], [22], [23]. Application require-

ments such as accuracy targets, throughput demands, and error tolerance can then guide lower-level decisions ranging from arithmetic approximations to execution-model selection and device operating conditions.

B. Outlook

The interactions outlined above suggest that many of the most impactful advances in digital memristive computing will emerge at the interfaces between abstraction layers rather than within isolated layers. First, automated cross-layer toolchains are needed to connect device characterization, circuit synthesis, architectural exploration, and workload evaluation within a unified design framework. Such methodologies can identify globally efficient design points and accelerate the development of future systems. Second, architectures should become increasingly paradigm-aware. Exact, approximate, and stochastic SL exhibit different requirements for communication, storage, reliability, and execution scheduling. Future architectures should therefore adapt resource organization and execution strategies to the targeted computing paradigm and workload characteristics. Third, software-centric methodologies will be essential for practical adoption. Compilers, runtimes, and programming abstractions must expose memristive computational resources through portable and accessible interfaces while accounting for communication, synchronization, and reliability constraints. Such abstractions will be critical for enabling scalable software development and broader adoption of digital IMC systems. Ultimately, translating advances in memristive devices into practical computing systems will require cross-layer methodologies that jointly optimize the full stack from materials and devices to architectures, software, and applications.

VI. Conclusion

Digital memristive in-memory computing has evolved from a device-centric concept into a computing paradigm that spans materials, devices, circuits, architectures, and workloads. Unlike conventional CMOS systems, SL directly exposes device behavior to computation, causing constraints and opportunities to propagate throughout the stack. This paper presented a cross-layer perspective on digital memristive IMC, highlighting how device technologies, computing paradigms, microarchitectures, and architectural organizations are fundamentally interconnected. We argued that exact SL, approximate SL, and SC represent complementary design points that navigate different trade-offs between correctness, efficiency, robustness, and complexity. The central message of this work is that future progress will increasingly depend on cross-layer co-design. Rather than optimizing individual layers in isolation, future research should focus on the interfaces between them, jointly considering device characteristics, logic execution, architectural organization, and application requirements. We believe this perspective provides a useful framework for understanding the current design space, identifying open challenges, and positioning future work in digital memristive IMC.

Acknowledgements

This work was supported by the German Research Foundation (DFG) under Project 550835853 and Germany's Excellence Strategy – 2082/2 – 390761711, U.S. National Science Foundation (NSF) grants 2329096 and 2543446, the UIUC Center for Advanced Semiconductor Chips with Accelerated Performance (an NSF IUCRC), the UIUC DREMES HYBRID Center, and Hector Stiftung.

References

[1] O. Mutlu *et al.* A Modern Primer on Processing in Memory. arXiv:2012.03112 [cs.AR], 2025.

[2] M. Lanza *et al.* The Growing Memristor Industry. *Nature*, 2025.

[3] J. B. Roldán *et al.* Variability in Resistive Memories. *Advanced Intelligent Systems*, 2023.

[4] J. Borghetti *et al.* 'Memristive' Switches Enable 'Stateful' Logic Operations via Material Implication. *Nature*, 2010.

[5] S. Gupta *et al.* FELIX: Fast and Energy-Efficient Logic in Memory. In *ICCAD*, 2018.

[6] A. Sebastian *et al.* Memory Devices and Applications for In-Memory Computing. *Nature Nanotechnology*, 2020.

[7] S. Kvatinsky *et al.* MAGIC—Memristor-Aided Logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2014.

[8] D. Ielmini and H.-S. P. Wong. In-memory computing with resistive switching devices. *Nature Electronics*, 1:333 – 343, 2018.

[9] F. Seiler *et al.* ATOMIC: Automatic Tool for Memristive IMPLY Based Circuit-Level Simulation and Validation. In *SMACD*, 2025.

[10] A. Bende *et al.* Device-to-Logic Variability Propagation in RRAM-Based Logic-in-Memory Architectures. *Neuromorphic Computing and Engineering*, 2026.

[11] G. Gulafshan *et al.* Realistic Behavioral Model for ReRAMs Capturing Non-Idealities. *Communications Materials*, 2025.

[12] F. Seiler and N. TaheriNejad. Approximate Computing in Memristive Processing-in-Array (PIA). *IEEE Design & Test*, 2026.

[13] F. Seiler and N. TaheriNejad. Accelerated Image Processing Through IMPLY-Based NoCarry Approximated Adders. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[14] J. P. C. de Lima *et al.* All-in-Memory Stochastic Computing Using ReRAM. In *DAC*, 2025.

[15] O. Leitersdorf *et al.* AritPIM: High-Throughput In-Memory Arithmetic. *IEEE Transactions on Emerging Topics in Computing*, 2023.

[16] M. S. Q. Truong *et al.* RACER: Bit-Pipelined Processing Using Resistive Memory. In *MICRO*, 2021.

[17] R. Dittmann *et al.* Nanoionic Memristive Phenomena in Metal Oxides: The Valence Change Mechanism. *Advances in Physics*, 2021.

[18] I. Valov *et al.* Electrochemical Metallization Memories—Fundamentals, Applications, Prospects. *Nanotechnology*, 2011.

[19] K. A. Campbell. Self-Directed Channel Memristor for High Temperature Operation. *Microelectronics Journal*, 2017.

[20] J. Aghassi-Hagmann *et al.* Fully Printable Oxide TFTs and Memristors on Rigid and Flexible Substrates With Excellent Integration Properties for Hybrid Systems. In *IEDM*. IEEE, 2025.

[21] B. Hoffer *et al.* Experimental Demonstration of Memristor-Aided Logic (MAGIC) Using Valence Change Memory (VCM). *IEEE Transactions on Electron Devices*, 2020.

[22] M. S. Q. Truong *et al.* Adapting the RACER Architecture to Integrate Improved In-ReRAM Logic Primitives. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2022.

[23] M. S. Q. Truong *et al.* The Memory Processing Unit: A Generalized Interface for End-to-End In-Memory Execution. In *HPCA*, 2026.

[24] N. Talati *et al.* mMPU—A Real Processing-in-Memory Architecture to Combat the von Neumann Bottleneck. In *Applications of Emerging Memory Technology: Beyond Storage*. Springer Singapore, 2020.

[25] R. Ben-Hur *et al.* SIMPLER MAGIC: Synthesis and Mapping of In-Memory Logic Executed in a Single Row to Improve Throughput. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[26] A. A. Khan *et al.* CINM (Cinnamon): A Compilation Infrastructure for Heterogeneous Compute In-Memory and Compute Near-Memory Paradigms. In *ASPLOS*, 2025.