Integrating Processing-In-Memory into HW/SW Co-Design for Automotive Embedded Systems

Zineb El Kacimi, Safae Dahmani, Oussama Elissati, Mouhcine Chami
Institut National des Postes et Télécommunications, STRS Lab., Rabat, Morocco

***Abstract*—Continuous expansion of using intelligent learning workloads in modern vehicles, leads Electronic Control Units (ECUs) to manage massive volumes of data while dealing with hard real-time constraints. That said, ECUs must operate under strict power budget due to limited battery capacity and other safety and functional requirements. We study in this research the feasibility of integrating Processing-In-Memory (PIM) approach into the hardware/software co-design process for the emerging ECU architectures. The idea here is to allocate data-centric tasks to in-memory compute units so that computation occurs directly where data is stored. The proposed approach will allow avoiding expensive data traffic, which will improve processing performance and reduce energy consumption. Our work introduces a conceptual framework that reconciles the PIM strengths with automotive requirements and introduces techniques from dynamic voltage scaling to smarter memory management, and is limited to a conceptual architectural proposal without experimental validation or quantitative evaluation. We end up discussing some crucial opportunities and open challenges arising from the implementation of PIM in next-generation automotive systems.**

***Keywords*—Electronic Control Units; energy consumption; hardware/software co-design; Processing-In-Memory; real-time constraints**

I. Introduction

Nowadays, automotive industry is growing very quickly towards smart electric vehicles, which bring together assisted and autonomous driving, high connectivity, and a wide range of intelligent features. With the increase in complexity and performance demands, These developments marked a shift in embedded systems, especially Electronic Control Units (ECUs), the main nervous system of modern cars. It is important to highlight that ECUs are responsible for managing features from real-time sensor operations to battery management and various advanced driver assistance systems (ADAS) among many other functions. With the increase of data-centric features, we are driving classical design approaches towards increased computational power and real-time accuracy.

A focus on hardware/software co-design approaches is a key point to deal with new embedded systems. It works on optimizing both hardware and software together to achieve better performance and cost-effectiveness across the whole system. This technique has already demonstrated results in ECU subsystems: engine control [1], security firewalls [2], and reconfigurable driver designs [3]. Yet, the major remaining limitations are performance bottleneck related to data movement, scaling across different ECU types, and low latency under strict power constraints.

In such a context, Processing-In-Memory (PIM) [4] offers a promising computing paradigm, which can transform how current data processing in embedded devices is conducted. By processing computation close to the memory components, PIM immediately relieves the typical processor-memory bottleneck, leading to a significant reduction in the amount of data transfer and energy consumption. This computing paradigm has been reported successfully in accelerating data-dense applications such as Artificial Intelligence (AI) inference, graph processing, and real-time signal analysis [5], [6].

The importance of the growing use of AI and deep learning in modern vehicles, as reviewed in [7], [8], also highlights the demand for new, disruptive architectural approaches to handle complex, data-driven tasks. Having shown capabilities in parallel data flows, latency reduction, and low power costs, PIM can provide a great value to automotive applications because they are performance and power-constrained applications in addition to data-intensive, especially with the increasing use of AI.

Integrating PIM into hardware/software co-design moves data-intensive work to where the data already resides, reducing transfer overhead and latency. In practice, during the hardware/software co-design partitioning step, workloads with memory-bound tasks run on PIM while the control-heavy tasks stay on the CPU. That leads to balanced resources utilization and helps improve real-time responsiveness. Existing literature shows limited exploration of PIM integration in the automotive domain. The systematic integration of PIM into ECU architectures is highly dependent on its compatibility with automotive standardized hardware/software co-design techniques, and the challenges that can be faced to meet the requirements of automotive safety-critical systems.

This study aims to introduce a conceptual framework that combines hardware/software co-design principles with emerging PIM architectures to support the development of next-generation ECUs.

This study is organized as follows: Section II reviews the state-of-the-art of the hardware/software co-design in automotive embedded systems, focusing on ECUs and identifying bottlenecks that PIM can alleviate. Section III presents an analysis of recent advancements in PIM architectures across different memory technologies, in addition to a proposal for PIM-enabled ECU architecture and techniques for its integration. Section IV details the motivation and pathways for integrating PIM into automotive ECU architectures. Section V discusses challenges and opportunities for integrating PIM methodology into automotive ECUs. Finally, Section VI concludes the study along-with a declaration on the use of generative AI in Section VII.

Although no experimental implementation is provided, this work lays out the foundation for future development, prototyping, and evaluation, and we are currently exploring comparative benchmarking using existing gem5-based simulation tools.

II. Hardware-Software Co-Design in Automotive Embedded Systems

A. The Need for Hardware/Software Co-design in Automotive Embedded Systems

Hardware/software co-design is a methodology in which both the hardware and the software are considered interdependent components of a system, aiming to optimize them concurrently in the design process. This collaborative process helps in the early evaluation of trade-offs between performance, power, cost, area, safety, and flexibility to form integrated systems that are optimized to meet specific requirements at minimal cost. Hardware/software co-design, as outlined by Shaout, El-Mousa and Mattar in their work on the heterogeneous embedded systems [9], is an approach that consist of several activities including system partitioning, in which functions are strategically allocated between hardware and software according to performance, cost, and power limitations. Then this division is further optimized in task allocation and scheduling, matching tasks to meet timing constraints and reduce system latency. In addition, hardware/software interface definition helps in establishing clear communication flows for building and ensuring the compatibility of the components, while performance modeling and simulation help to identify the potential bottlenecks early on the design phase. Also, design space exploration is further an important step in which various levels of analysis can be made that allows for the trade-offs between competing needs (performance versus power consumption). Co-synthesis and co-verification guarantee accurate joint behavior, so hardware and software are developed and tested in parallel. Finally, integration and validation are the last steps, enabling the testing of the entire system to determine whether it meets its design specifications.

In sectors, where the rate of technological developments is accelerating, such as automotive, it becomes even more pressing to adopt such a method. Given the increasing adoption of embedded computing systems used in cars, hardware/software co-design is becoming an apparent need. A case study of how this method can expedite computationally intensive tasks, and be used to accelerate computation has been reported in embedded security, with the Elliptic Curve Cryptography accelerator on a Zynq SoC producing impressive efficiency gains based on an efficient hardware/software partitioning [10]. Real-time performance, safety, power efficiency, and cybersecurity are now fundamental requirements in modern vehicles, and hardware/software co-design ensures that computing architectures are optimized to meet these constraints while supporting the integration of evolving components and secure connectivity, as highlighted by Abdallah et al. [11].

Integrating PIM into hardware/software co-design can create new opportunities to improve the balance of performance and power efficiency. Thus, PIM also reduces overall overheads and latencies caused by data transfer by directly conducting data-heavy computations in memory, which helps to optimize system partitioning and resource allocation. Inherently, hardware/software co-design frameworks need to consider the computational capacity of the PIM-enabled memory and allocate tasks such as sensor-centric data handling to memory, whereas other high-level control functions should be retained by the CPU. Task scheduling can take further advantage of the parallel data flows generated by PIM and reduce latency and energy usage.

Given the increasing complexity of embedded computing systems in modern vehicles, we are seeing a rise in the number of ECUs, each packed with multiple software features, which leads to an obvious need for updating hardware/software co-design process to meet the new design requirements and ensure both reliability and performance. In [12], Axelsson points out that to address these challenges, hardware/software co-design is needed across the overall system architecture, not only at the ECU level.

Hardware/software co-design has moved from being a "nice-to-have" best practice to a core engineering discipline, therefore, adopting advanced co-design methods is essential to balance today's competing priorities including performance, cost, safety, and power.

Current hardware/software co-design strategies provide different solutions for varying ECU types, each with its own trade-offs. We will review these approaches to point out current weaknesses, particularly for handling real-time performance of data extensive tasks, scalability and power efficiency. This analysis provides the foundation for evaluating the architectural shift towards integrating PIM to address these core limitations.

B. ECU: The Heart of Automotive Electronics

An ECU is a specialized embedded system, which controls various functions in a larger system, mainly through sensor inputs, control algorithms, and actuator operation. Modern automotive systems have very high use cases of ECUs that make them perform from basic tasks to high-end processes. Automobile ECUs were originally single-unit components: a microcontroller, sensors, and actuators connected via point-to-point wiring with other vehicle components. This simple architecture was acceptable for early vehicle designs with minimal electronic requirements (see Fig. 1). For instance, a Powertrain Control Unit (PCU) manages engine and transmission functions, ensuring optimal fuel efficiency and emissions compliance. Meanwhile, a Body Control Module (BCM) handles auxiliary functions such as lighting and climate management. For the driver and passengers, infotainment ECUs consolidate all multimedia and connectivity functions. However, the most computationally intensive tasks are computed by ADAS systems ECUs, which receive real-time input from cameras, radar, and LiDAR to perform features like adaptive cruise control and collision avoidance. These cases demonstrate that ECUs support everything in the modern automobile from its most basic operations to its most advanced autonomous functions.

It's clear that today's vehicles come up with significantly more complex needs compared to initial automotive requirements. The current automotive environment has set new standards for ECU design [13]. In fact, modern architectures go beyond standalone microcontroller units and evolve toward more integration of domain controllers and centralized computing platforms [14], [15].

The most obvious challenge in these architectures is the incorporation of workloads which demand considerable computational power to process large volumes of sensor data in real-time. Traditional ECUs were designed primarily for the purpose of relatively simple control applications, but for these demanding applications often lack both the processing capacity and architectural flexibility needed to support such demanding applications. Thus, to handle the computationally intensive tasks of applications like sensor fusion and real-time control, current designs integrate dedicated hardware accelerators. This hybrid architectural evolution reflects the industry's adaptation to growing functional complexity and performance demands.

In the next subsection, we will explore the existing hardware-software co-design methodologies to highlight opportunities for further optimization, which leads us to novel approaches such as PIM to address the challenges of next-generation ECU design.

C. Existing Hardware/Software Co-design Methodologies for ECUs

Various hardware-software co-design strategies have been explored in the area of ECUs, one of the most indispensable components of modern automotive systems. The goal of these hardware/software co-design methods is to find the optimal split of tasks between hardware and software, incorporating the best of both worlds to satisfy the tight performance, power, and security specifications. For instance, hardware/software partitioning of crankshaft functionality on an FPGA delivered a 58% performance improvement [1]. This demonstrates co-design's ability to optimize critical automotive functions. Yet, a significant bottleneck is scalability; extending this approach to other ECUs is indeed a significant challenge. Coupled with the high cost of FPGA-based solutions that may lead to a lack of widespread industry adoption.

In the context of security domain, Pesé et al. [2] introduced an embedded firewall combined of hardware and software to secure in-vehicle networks, such as Automotive Ethernet, this firewall has been shown to minimize latency and jitter while maintaining constant throughput under dynamic conditions. Nevertheless, integration issues and resource limitations restrict this system's applicability across broader automotive systems. Another subsequent study [3] proposes a reconfigurable ECU architecture that co-integrates security and dependability primitives to address the critical need for security without sacrificing performance, reporting a 47.9×speedup and 2.4×better energy efficiency compared to a multi-core automotive processor. This work provides quantifiable evidence that hardware/software co-design can successfully overcome the traditional performance versus security trade-off.

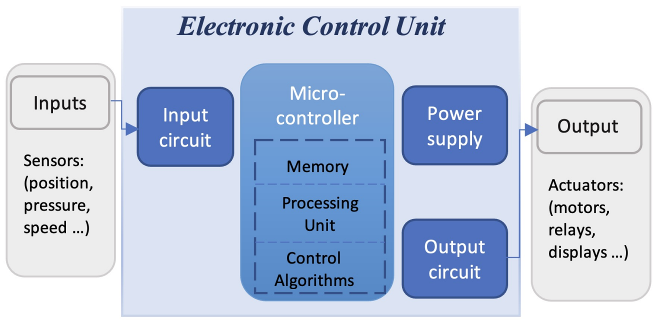

Fig. 1. Simplified architecture of early ECUs with basic microcontroller-based designs.

This principle of embedding efficient, transparent security is further demonstrated by a lightweight FPGA-based IDS-ECU for the CAN bus [16]. The architecture introduces a hardware-accelerated intrusion detection system that operates transparently to the primary ECU software. Compared to GPU-based implementations, it achieved a 2.3×reduction in per-message latency and a 15×reduction in power consumption. This shows that integrating built-in operating capabilities, such as security monitoring, into ECUs not only doesn't compromise its performance but can even improve it. FPGA-based remote reconfigurable ECUs suggested in [17] offer flexibility and adaptability but struggle in ensuring secure remote updates and managing implementation costs. Another hardware-software co-design approach was developed in [18] for an automotive application, a Multiple Injection Driver Automotive Subsystem. This approach consists of a configurable system-on-chip (CSoC), which consists of a microprocessor and embedded FPGA. Implementing this technique enabled rapid system design cycles, promoting component re-use, and working within tight timelines for complex embedded system. However, the authors note that a key limitation of their specific flow was that the interface synthesis was not fully automatic, requiring some minimal manual steps, which presents a challenge for complete design automation. While these studies have shown considerable promise in optimizing ECU performance, power consumption and security, modern automotive systems are incorporating increasingly data-intensive applications that further introduce the bottleneck of data movement, which leads to higher power consumption and limits real-time processing. To address these challenges, PIM can complement hardware/software co-design methodologies by offloading data-intensive tasks from traditional processors, and this can also ensure robust protection against dynamic threats by accelerating intrusion detection algorithms and firewall operations, in the context of automotive security applications.

The subsequent section will discuss current research on PIM technology, including its underlying mechanisms, and examine its potential to tackle challenges in modern automotive systems.

III. Overview of PIM and Related Works

PIM is a modern computing paradigm that integrates computational capabilities in or near memory systems. This approach was introduced to mitigate the performance limitations of the traditional Von Neumann architecture. In the latter, processors are physically separated from memory, causing heavy traffic between them. Significant latency and energy overhead are caused by memory-processor bottlenecks. This is particularly true for intensive data workloads which is the case for intelligent learning, graph processing or sensor fusion computations in automotive. By putting processing close to where the data is stored, PIM minimizes data movement, reduces network related latency, in addition to important energy savings.

The processor-memory bottleneck is a big challenge for modern computing systems. For instance, [19] states that traditional architectures allocate up to 99.75% of their energy consumption in floating-point operations on memory access rather than computation itself. PIM solves this problem by fitting processing units directly into existing memory modules to enable operations, like matrix multiplications, filtering, and pattern recognition, to be done in place. This makes PIM a good match for AI-driven tasks.

A. Overview of PIM Architectures

The performance of PIM architectures heavily depends on the developments in memory technologies, design approaches, and application fields. These factors provide the foundation for integrating computational capabilities directly within memory systems. In the following, we describe the important characteristics of PIM by design type, application field, and memory technology, according to Asifuzzaman et al. in [5].

1) PIM by design type: PIM architecture can be classified according to its computational features integrated into or near memory system. Table I provides descriptions and features of the design types.

2) PIM by application field: There are several application domains where PIM designs help to achieve maximum efficiency. Table II illustrates the major application fields and their use cases.

3) PIM by memory technology: Memory technologies are a major factor for effective PIM implementation. Table III presents the common memory technologies and their suitability for PIM.

The presented classifications highlight diverse PIM-based approaches, addressing conventional memory limitations considering emerging applications' demands. In the next subsection, we investigate how developments in PIM bring transformational performance and energy efficiency within different sectors.

B. Advancements in PIM Architectures

In this section, we present a wide range of examples of PIM-based architectures. Associative In-Memory Processors (AiMP), which are discussed in [20], utilize resistive memory for direct access to memory arrays, improving throughput and energy consumption in clustering and matrix operations. In [21], the authors demonstrate that hybrid schemes like Hetraph, are able to combine analog and digital processors, and this combination tradeoff for performance and energy efficiency for irregular workloads such as graph processing. Hetraph provides up to 7.54×speedup over CPUs, and 1.56×over GPUs, resulting in significant energy savings of up to 57.58×vs CPUs, 19.93×vs GPUs.

Beyond recent hardware innovations, software approaches have been developed to fully utilize the potential of PIM. In this regard, for example, the work in [19] presents a hardware/software co-design methodology, which specifically targets associative in-memory processors and employs dynamic voltage scaling techniques and selective instruction scheduling. On average, the proposed method achieves a 7% decrease in memory area and an 18% reduction in overall energy consumption.

Moreover, the PIM architectures programming frameworks focus on task division between control-intensive (CPU) and data-intensive (PIM) components, avoiding unnecessary information exchanges between memory and processors. Similar paradigms have been applied to IoT systems, as demonstrated in [22], where they improve data processing time and power consumption, leading to enhanced utilization efficiency. In addition, J. Joo et al. [23] examine the utilization of PIM-based systems for regular expression matching and report an up to 96% reduction in execution time in comparison to legacy multithreaded implementation methods. Another interesting work [6] introduced the TransPIM that integrates processing capabilities in its memory for transformer workloads. It demonstrates performance gains of 22.1×to 114.9×compared to GPUs. Its token-based processing approach minimizes data movement and enhances scalability for long sequences, making it suitable for real-time applications like sensor fusion. Similarly, the iMTransformer, presented in [24], utilizes PIM technologies to accelerate attention mechanisms in transformer networks with 13.71×speedup and 12.57×better energy efficiency compared to GPUs. Another approach presented in [25] advances in memory Big Data processing by combining efficient indexing (e.g., B trees, R trees), parallelism (scale up/scale out in systems like Apache Spark), and memory optimization (e.g., RowClone). These techniques reduce latency and improve efficiency by executing bulk operations near memory.

Due to its previously discussed capacity to improve energy consumption, latency and throughput, PIM technology offers an effective solution to address the growing data-centric challenges in automotive systems, a domain that is rapidly evolving toward electric, connected and autonomous vehicles.

IV. Motivation and Pathways for Integrating PIM into Automotive ECU Architectures

The ECUs have been heavily demanded by the rapid technological change in automotive systems, including autonomous driving, ADAS, and electric vehicles (EVs). Given the need for high-throughput and real-time sensor data such as LiDAR, radar, cameras, etc., embedded systems must maintain specific constraints including latency, energy consumption, safety, and cost. At the same time, traditional Von Neumann architectures introduce bottlenecks caused by intensive processor-memory data movement, which is associated with high energy consumption, latency, and poor scalability, and this is limiting the ability of ECUs to meet the growing complexity of modern automotive systems.

A. Unlocking the Potential of PIM for Automotive Applications

Below, we provide a table (see Table IV) for the key advantages of PIM for automotive systems. The table also gives some examples of ECU applications best optimizing the given advantages.

TABLE I. TYPES OF PIM DESIGNS

Design Type	Description	Examples/Key Features
Cell-level PIM	Simple operations performed at the cell level of the memory array.	Basic arithmetic or logical operations within memory cells.
Sense amplifier / row-buffer PIM	More complex operations performed at the sense amplifier or row-buffer level.	Suitable for operations requiring faster access to larger memory segments.
Near-memory PIM	Processing cores placed near memory banks to execute a subset of CPU instructions.	Enables larger-scale computations and supports diverse workloads.
Hybrid PIM	Combines multiple PIM approaches (e.g., analog and digital processing) to optimize performance.	Supports irregular workloads and balances efficiency for diverse applications.

TABLE II. PIM APPLICATION FIELDS AND THEIR KEY USE CASES

Application Field	Description	Examples/Use Cases
Deep Neural Networks (DNNs)	Optimized for compute patterns of DNNs using specialized memory technologies like ReRAM.	Matrix multiplications, convolution operations, and attention mechanisms in transformers.
Graph Processing	Designed for irregular workloads and high data movement requirements.	Algorithms like BFS, PageRank, and graph clustering.
AI and Machine Learning	Accelerates AI workloads with in-memory computations to reduce latency and energy consumption.	Transformer workloads, pattern recognition, and filtering tasks.
IoT and Edge Computing	Enhances data processing efficiency for real-time applications in IoT systems.	Sensor fusion, real-time analytics, and distributed systems.

TABLE III. COMPARISON OF MEMORY TECHNOLOGIES SUITABLE FOR PIM

Memory Technology	Description	Suitability for PIM
DRAM	Widely used commodity memory technology with affordable costs.	Limited scalability but suitable for general-purpose PIM designs.
HBM (High-Bandwidth Memory)	3D-stacked DRAM variant offering high bandwidth and efficiency.	Ideal for data-intensive applications requiring fast memory access.
HMC (Hybrid Memory Cube)	3D-stacked memory integrating DRAM layers with a logic base layer.	Reduces latency and increases bandwidth for high-performance computing.
STT-MRAM	Non-volatile memory using magnetic properties to store data.	High endurance and fast access speeds make it suitable for frequent data access.
ReRAM	Non-volatile memory using resistive switching for data storage.	Performs computations directly within its unique data array.
PCM (Phase Change Memory)	Relies on material phase changes for data storage, offering high density and scalability.	Suitable for large-scale data storage and processing.

TABLE IV. WHY PIM IS ESSENTIAL FOR AUTOMOTIVE SYSTEMS

Key Benefit	Brief Description	Example ECU/System
Reduction in Data Movement	By performing computations directly in memory, PIM minimizes data transfer between processors and memory, reducing latency and energy consumption.	ADAS ECUs: Real-time sensor fusion for lane detection and collision avoidance.
Improved Energy Efficiency	PIM reduces energy usage by eliminating unnecessary data movement and optimizing memory operations, making it ideal for energy-constrained systems.	Battery Management Systems: Efficient monitoring and optimization of EV battery performance.
Scalability for Complex Workloads	PIM architectures can handle the growing complexity of workloads, such as neural network inference and multi-sensor fusion, without performance bottlenecks.	Autonomous Driving Systems: Path planning and obstacle detection using data from cameras, radar, and LiDAR.
Real-Time Performance	PIM accelerates time-sensitive computations, ensuring ECUs can meet stringent latency requirements in critical systems.	Braking and Steering ECUs: Fast decision-making for ABS and electronic steering control.

B. Techniques for Integrating PIM into ECU Architectures

1) Dynamic Voltage Scaling (DVS): As shown by Yantir et al. [19], can be effectively integrated into hardware/software co-design to optimize energy efficiency and responsiveness in PIM-enhanced subsystems like Powertrain or Braking Control. Dynamic Voltage Scaling is a co-design methodology realized by combining the need for tight coupling of hardware configurability with software control. Multi-VDD designs enable the dynamic adjustment of memory bank voltage in response to computational load at the hardware level. Simultaneously at the software level, a scheduler is used to align the execution of the task with these power states. To illustrate, during steady cruising voltage may be decreased for saving power, rise for acceleration or scale with braking intensity to maintain safety and fulfil real-time computational demands. Through the coupling of hardware voltage scaling with software task scheduling, DVS guarantees an effective operation without affecting system responses in mission-essential automotive applications.

2) Hierarchical memory management: Divides data into memory tiers; mission-critical tasks can be performed within the rapid SRAM, while less urgent tasks within slow DRAM or STT-RAM to balance latency and efficiency. PIM, along with an optimized hardware/software co-design, improves this by allowing computations to run within the right memory tier with reduced distance to data. For instance, brake and steering ECUs run on PIM-based SRAM for real-time, low-latency processing, and infotainment programs to perform PIM tasks in slow-tier memory to free faster memory. Research on STT-RAM caches [26] shows that heterogeneous memory designs can optimize retention time and energy, and hierarchical PIM therefore is well-suited to different automotive workloads. Complementing this, the study on Big Data architecture, presented in [27], highlights the importance of a distinct memory management layer, which automatically caches frequently used data and optimizes data locality. This concept aligns with hierarchical memory management in its goal of improving performance by minimizing data movement and ensuring that critical tasks are stored and executed in the most appropriate memory tier.

3) Sensor-specific processing: Uses token-based dataflows that allow individual sensors, including cameras, radar, and LiDAR, to process their data independently in PIM-enabled memory and in compliance with hardware/software co-design principles. This technique, inspired by the TransPIM architecture [6], exploits a token-based processing protocol to manage sensor workloads through local processing of data tokens in memory. Every token, representing a unit of information based on the sensor data including for instance, lane detection from cameras or speed from radar, is processed directly in memory, thus reducing data movement and latency. Building on this work, hardware/software co-design frameworks facilitate system partitioning to distribute computation-intensive operations to PIM near the sensor, while software algorithms dynamically coordinate job scheduling and data aggregation among sensors on-the-fly. The main advantage of this approach is parallelizing sensor data processing in memory through concurrent scheduling of the workload and optimal distribution of tasks. Such an integration can then, minimize latency and improve performance for real-time applications such as ADAS and autonomous driving.

4) Task Partitioning between CPU and PIM: Aligns with hardware/software co-design principles, ensuring efficient resource utilization by prioritizing workloads according to their nature. On one side, you have control-intensive tasks, such as temperature sampling and adjusting speeds, are expected to be done on the CPU. On the other hand, data-driven tasks like data consolidation for sensor data shall be computed in memory for faster processing. Hardware/software co-design frameworks allow this partitioning by ensuring that the right tasks can be dynamically assigned to the right resource. The latter structure works well for ADAS and allows PIM accelerators to perform real-time image recognition, while safety-critical control operations such as braking and steering modifications are run simultaneously on the CPU. Just like battery management systems in EVs can use PIM to analyze sensor data while control functions, like adjusting the charging rate, remain on the CPU.

C. Proposed PIM-Enabled ECU Architecture

A main approach for integrating PIM within automotive systems is to optimize a single ECU by embedding processing capabilities directly into its memory subsystem. As an example, we choose a collision avoidance ECU (see Fig. 2); the system integrates a CPU and a PIM-enabled memory module to handle critical data processing tasks. The PIM units can perform early-stage computations on sensor inputs, such as radar and camera data, and execute functions like object detection, distance estimation, as well as threat classification directly in memory, while the CPU will take care of high-level decisions and control logic. Hence, more complex systems can be built using this architecture.

Following this, the architecture can be expanded to a more complex system, which is derived from the Domain Controller ECU paradigm presented in [28]. It promotes the integration of multiple distributed ECUs into a centralized framework. It allows for real-time processing, simplified maintenance, and over-the-air (OTA) software updates. In that concept, diverse domain-specific tasks are performed in a single central ECU to effectively serve as the system's command center. Taking this paradigm and applying it to the ECU system, our proposed architecture works towards implementing PIM capabilities to offload and accelerate data-intensive tasks.

Fig. 3 illustrates the proposed conceptual integration of PIM into a centralized automotive ECU architecture. The system is organized around a central ECU platform responsible for high-level control, task scheduling, and coordination. Based on intelligent workload partitioning, the highly parallel tasks are offloaded from the central processor to a shared PIM-enabled memory subsystem.

However, certain sensor data requiring immediate high processing is sent directly to the central CPU before being forwarded to the PIM subsystem for data-heavy computations.

This subsystem consists of specialized PIM blocks, each associated with a different sensor domain, such as camera, radar, or LiDAR, and capable of performing early-stage, in-memory processing. Tasks like AI-based image filtering, Doppler speed estimation, and neural network inference are executed directly within the memory units. The output from these PIM units is stored in a hierarchical memory structure that uses fast SRAM for time critical control loops and non-volatile STT-RAM or DRAM for storing historical or less critical data. The information collected is preprocessed and then sent to the central CPU, where high-level decisions are made based on this refined data. Then, the CPU generates control commands that are forwarded directly to the actuators, such as those responsible for advanced driver-assistance systems (ADAS). In

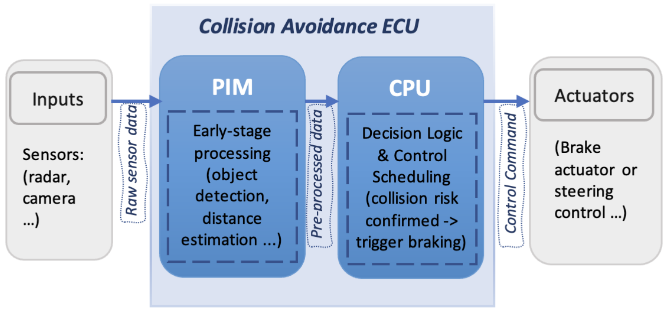
Collision Avoidance ECU
Inputs
Sensors: (radar, camera ...)
Raw sensor data
PIM
Early-stage processing (object detection, distance estimation ...)
Pre-processed data
CPU
Decision Logic & Control Scheduling (collision risk confirmed -> trigger braking)
Control Command
Actuators
(Brake actuator or steering control ...)

Fig. 2. Architecture of PIM integrated in collision avoidance ECU.

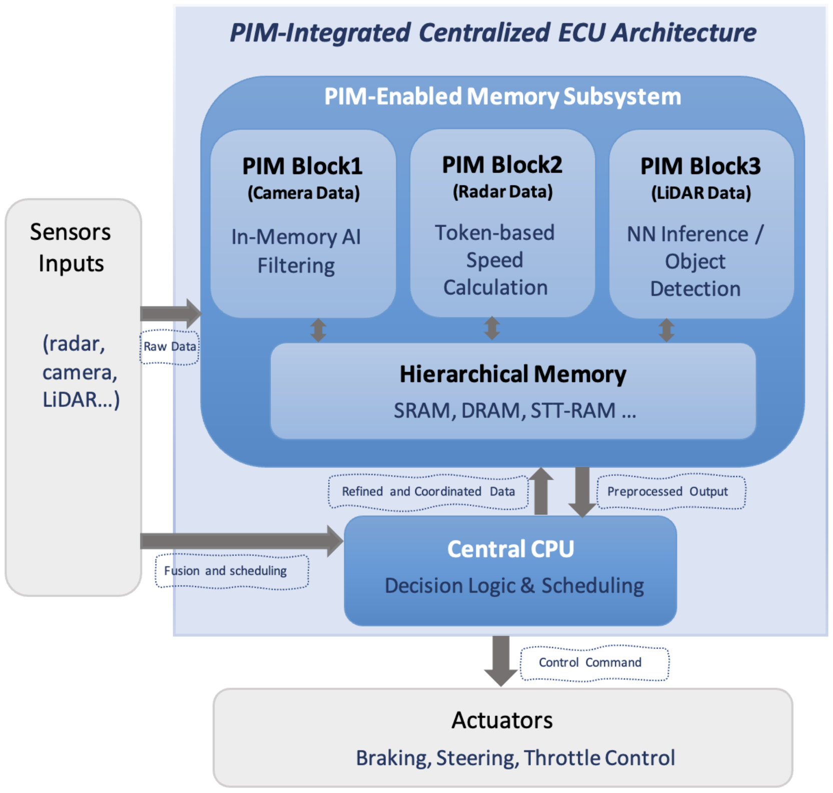
PIM-Integrated Centralized ECU Architecture
PIM-Enabled Memory Subsystem
PIM Block1
(Camera Data)
In-Memory AI Filtering
PIM Block2
(Radar Data)
Token-based Speed Calculation
PIM Block3
(LiDAR Data)
NN Inference / Object Detection
Hierarchical Memory
SRAM, DRAM, STT-RAM ...
Sensors Inputs
(radar, camera, LiDAR...)
Raw Data
Refined and Coordinated Data
Preprocessed Output
Fusion and scheduling
Central CPU
Decision Logic & Scheduling
Control Command
Actuators
Braking, Steering, Throttle Control

Fig. 3. PIM centralized ECU architecture.

PIM architectures, intelligent processing is distributed across different blocks of the memory subsystem and the CPU. Real-time responsiveness is then enhanced thanks to the reduction of memory traffic and redundant computations.

To conclude, the PIM-based hybrid architecture is showing promising performance and energy efficiency improvements in the context of emerging complex applications.

V. Challenges and Opportunities

Integrating PIM into automotive ECU architectures is not an easy task. Several critical challenges are identified to create successful PIM-based heterogeneous systems. Automotive ECUs from multiple generations communicate over deterministic networks, such as CAN, LIN, and FlexRay. Hence, PIM components, as new nodes, must remain compatible with these legacy buses and existing communication stacks. Second, the absence of standardized programming paradigms and compiler support for PIM limits its accessibility for embedded developers. Here are several aspects of potential solutions that still need development: high-level APIs, task schedulers, hardware/software interfaces that facilitate the adoption of PIM in automotive software stacks, particularly under AUTOSAR or embedded Linux environments. Additionally, automotive applications require deterministic timing behavior to satisfy real-time constraints, and PIM will introduce new execution patterns that will further complicate latency and worst-case execution time (WCET) analysis, while these analysis are crucial for real-time safety-critical tasks. From a security perspective, PIM-based systems need to maintain robust security measures to prevent system level threats such as malicious data injection or memory-side channel attacks. Moreover, fault isolation between PIM and CPU elements is necessary to avoid failures spreading across critical functions.

VI. Conclusion

A conceptual framework for implementing a PIM model within automotive ECU architectures is proposed in response to some of the core challenges in embedded systems, such as power consumption reduction, data movement avoidance, and real-time processing support needed by advanced automotive applications like ADAS, autonomous driving, and intelligent battery management. Building on existing hardware/software co-design methodologies and the latest developments in PIM architectures, the new processing framework that integrates PIM through a refined hardware/software co-design methodology to effectively deploy PIM within the strict constraints of safety-critical and resource-limited automotive environments. This contribution remains conceptual and does not include experimental validation or modeling.

Future work will focus on validating the concept through detailed simulation using cycle-accurate architectural simulators such as gem5, DRAMSim3, or PIMEval. The evaluation process will include metrics such as latency, energy consumption, memory traffic reduction, and worst-case execution time (WCET). Representative automotive workloads, such as ADAS perception pipelines, radar signal processing, and battery management algorithms, will be ported to PIM-based execution models.

To sum up, through this work we established a foundation for real-time and safety-critical use of PIM technologies in the context of hardware/software co-design of data-intensive automotive systems to improve performance and power efficiency.

VII. Declaration On Generative AI

During the preparation of this manuscript, the authors utilized Gemini and ChatGPT to support drafting and language refinement, but the development of scientific ideas, arguments, results, and conclusions was carried out only by human authors. The authors take full responsibility for the final version of the manuscript.

References

[1] Y. K. Chein, M. Fariq, F. Rahman, N. Kamsani, and F. Rokhani, "Hardware software partitioning of crankshaft function in engine control units using fpga-based testing," in *Proceedings of the IEEE 15th Student Conference on Research and Development (SCOReD)*, 2017, pp. 131–136.

[2] M. D. Pesé, K. Schmidt, and H. Zweck, "Hardware/software co-design of an automotive embedded firewall," *SAE Technical Paper*, 2017.

[3] B. Poudel and A. Munir, "Design and evaluation of a reconfigurable ecu architecture for secure and dependable automotive cps," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 1, pp. 235–252, 2018.

[4] X. Zou, S. Xu, X. Chen, L. Yan, and Y. Han, "Breaking the von neumann bottleneck: architecture-level processing-in-memory technology," *Science China Information Sciences*, vol. 64, no. 6, p. 160404, 2021.

[5] K. Asifuzzaman, N. R. Miniskar, A. R. Young, F. Liu, and J. S. Vetter, "A survey on processing-in-memory techniques: Advances and challenges," *Memories-Materials, Devices, Circuits and Systems*, vol. 4, p. 100022, 2023.

[6] M. Zhou, W. Xu, J. Kang, and T. Rosing, "TransPIM: A memory-based acceleration via software-hardware co-design for transformer," in *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 1071–1085.

[7] M. A. Jabu, A. A. Alugongo, and N. Z. Nkomo, "A critical review on the use of artificial intelligence in the automotive industry," *International Journal of Engineering Trends and Technology*, vol. 73, no. 6, pp. 450–456, 2025.

[8] S. O. Laghzal and A. EL OUADI, "Mapping artificial intelligence research in automotive manufacturing: A bibliometric study," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 16, no. 11, pp. 923–934, 2025.

[9] A. Shaout, A. H. El-Mousa, and K. Mattar, "Specification and modeling of hw/sw co-design for heterogeneous embedded systems," in *Proceedings of the World Congress on Engineering*, vol. 1, 2009.

[10] K. V. Patel, M. V. Shah, P. P. Prajapati, and A. J. Kshatriya, "Design and implementation of effective elliptic curve cryptography accelerator using hardware/software co-design on zynq board," *International Journal of Engineering Trends and Technology*, vol. 70, no. 8, pp. 327–335, 2022.

[11] A. Abdallah, E. M. Feron, G. Hellestrand, P. Koopman, and M. Wolf, "Hardware/software codesign of aerospace and automotive systems," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 584–602, 2010.

[12] J. Axelsson, "Hw/sw codesign for automotive applications: Challenges on the architecture level," in *Proceedings of the Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, 2001, pp. 123–126.

[13] H. Askaripoor, M. Hashemi Farzaneh, and A. Knoll, "E/e architecture synthesis: Challenges and technologies," *Electronics*, vol. 11, no. 4, p. 518, 2022.

[14] S. Sinha, "Towards a centralized multicore automotive system," Ph.D. dissertation, Boston University, Boston, MA, USA, 2022.

[15] S. Sinha, A. Farrukh, and R. West, "ModelMap: A model-based multi-domain application framework for centralized automotive systems," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[16] S. Khandelwal and S. Shreejith, "A lightweight fpga-based ids-ecu architecture for automotive can," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (ICFPT)*, 2022, pp. 1–9.

[17] K. Cho, J. Kim, D. Y. Choi, Y. H. Yoon, J. H. Oh, and S. E. Lee, "An fpga-based ecu for remote reconfiguration in automotive systems," *Micromachines*, vol. 12, no. 11, p. 1309, 2021.

[18] M. Baleani, M. Conti, A. Ferrari, and A. Sangiovanni-Vincentelli, "Hw/sw codesign of a multiple injection driver automotive subsystem using a configurable system-on-chip," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2002, pp. 87–92.

[19] H. E. Yantır, A. M. Eltawil, and K. N. Salama, "A hardware/software co-design methodology for in-memory processors," *Journal of Parallel and Distributed Computing*, vol. 161, pp. 63–71, 2022.

[20] L. Yavits, "Will computing in memory become a new dawn of associative processors?" *Elsevier*, 2023.

[21] Y. Huang, L. Zheng, P. Yao, J. Zhao, X. Liao, H. Jin, and J. Xue, "A heterogeneous pim hardware-software co-design for energy-efficient graph processing," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 684–695.

[22] X. Yang, Y. Hou, and H. He, "A processing-in-memory architecture programming paradigm for wireless internet-of-things applications," *Sensors*, vol. 19, no. 1, p. 140, 2019.

[23] J. Joo, H. Kim, H. Han, E. G. Im, and S. Kang, "Highly parallel regular expression matching using a real processing-in-memory system," *IEEE Access*, vol. 13, pp. 18 937–18 951, 2025.

[24] A. F. Laguna, M. M. Sharifi, A. Kazemi, X. Yin, M. Niemier, and X. S. Hu, "Hardware-software co-design of an in-memory transformer network accelerator," *Frontiers in Electronics*, vol. 3, p. 847069, 2022.

[25] B. H. Malik, M. Maryam, M. Khalid, J. Khlaid, N. U. Rehman, S. I. Sajjad, and et al., "Fast and efficient in-memory big data processing," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 10, no. 5, pp. 517–522, 2019.

[26] D. Gajaria, K. A. Gomez, and T. Adegbija, "STT-RAM-based hierarchical in-memory computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 9, pp. 1615–1629, 2024.

[27] M. Dessokey, S. M. Saif, H. Eldeeb, S. Salem, and E. Saad, "Importance of memory management layer in big data architecture," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 13, no. 5, pp. 460–466, 2022.

[28] D. Wang and S. Ganesan, "Automotive domain controller," in *Proceedings of the International Conference on Computing and Information Technology (ICCIT)*, 2020, pp. 1–5.