JOURNAL OF INTERNATIONAL CRISIS AND RISK COMMUNICATION RESEARCH
ISSN: 2576-0017
2025, VOL 8, NO S12

Understanding Memory Hierarchies In Embedded AI: From Cache To DDR

Ishan Pardesi

Carnegie Mellon University, USA.

Abstract
The evolution of artificial intelligence applications within embedded systems has fundamentally altered memory architecture design paradigms, necessitating sophisticated approaches to manage complex hierarchies spanning cache systems, tightly coupled memory, and external DRAM controllers. Contemporary embedded AI processors must balance competing demands of access latency, bandwidth capacity, power consumption, and deterministic timing constraints across heterogeneous computing architectures integrating CPU cores, graphics processors, and specialized accelerators. Energy disparities between on-chip and off-chip memory access drive optimization strategies, including hierarchical data reuse, cache-aware tensor placement, and deterministic memory allocation schemes. Model quantization techniques reduce memory footprint and bandwidth requirements while maintaining inference accuracy, enabling deployment of large neural networks on resource-constrained platforms. Quality-of-service arbitration mechanisms and advanced memory controller configurations prevent bandwidth starvation in multi-agent systems executing concurrent AI workloads. The synthesis of cache optimization, tightly coupled memory utilization, external memory controller tuning, quantization strategies, and zero-copy data flow establishes a comprehensive framework for embedded AI system design, enabling substantial throughput improvements and power consumption reductions while meeting real-time safety-critical requirements.

Keywords: Memory Hierarchy Optimization, Embedded AI Systems, Cache Coherency Protocols, Tightly Coupled Memory, Neural Network Quantization.

1. Introduction

The proliferation of artificial intelligence applications in embedded systems has fundamentally transformed the requirements and constraints of memory architecture design. Contemporary embedded AI processors must navigate increasingly complex memory hierarchies that span multiple orders of magnitude in access latency, bandwidth capacity, and power consumption. Research by Sze et al. demonstrates that the energy cost of data movement in deep neural networks significantly exceeds the energy cost of computation, with DRAM access consuming approximately 200 picojoules compared to 0.9 picojoules for a 32-bit integer addition [1]. This energy disparity underscores why memory hierarchy optimization represents a critical determinant of inference performance, energy efficiency, and economic viability across application domains ranging from mobile computing to automotive safety systems.

The challenge facing embedded systems engineers lies not merely in understanding individual memory components but in comprehending how these elements interact within heterogeneous computing architectures. Modern systems-on-chip (SoCs) integrate multiple processing elements—CPU cores, graphics processors, and specialized AI accelerators—each with distinct memory access patterns and bandwidth requirements. Quantitative analysis reveals that memory bandwidth requirements for neural

network inference can reach 100-200 GB/s for real-time video processing applications, with convolutional layers demanding sustained data transfer rates that stress the capabilities of DDR4-3200 memory controllers operating at the theoretical maximum bandwidth of 25.6 GB/s per channel [1]. The memory hierarchy must simultaneously accommodate the deterministic latency requirements of real-time control loops, the sustained throughput demands of neural network inference, and the burst access patterns of sensor data acquisition.

The ARM Cortex-A78 processor architecture exemplifies modern approaches to memory hierarchy design for AI workloads, implementing a sophisticated three-level cache structure optimized for computational efficiency. The Cortex-A78 features separate L1 instruction and data caches of 64 KB each with 4-way set associativity, delivering single-cycle access latency for cache hits [2]. The L2 cache extends from 128 KB to 512 KB with configurable associativity, providing intermediate storage that reduces L3 and external memory access frequency [2]. This hierarchical organization enables the processor to achieve sustained memory bandwidth sufficient for AI inference while minimizing power consumption, as on-chip cache accesses consume orders of magnitude less energy than external DRAM transactions [1].

This analysis examines the architectural principles governing memory hierarchies in embedded AI systems, with particular emphasis on the relationship between hardware design choices and inference workload characteristics. The discussion progresses from the smallest, fastest memory elements near processing cores through to external memory controllers managing gigabytes of system DRAM. Empirical studies demonstrate that optimized data reuse strategies can reduce external memory bandwidth requirements by 10× to 100× for convolutional neural networks, translating directly to proportional energy savings given the 200:1 energy ratio between DRAM access and on-chip computation [1]. Throughout this examination, specific attention is directed toward the quantifiable performance implications, power consumption trade-offs, and design constraints that distinguish embedded AI systems from traditional computing architectures.

Table 1: Memory Hierarchy Characteristics in Embedded AI Processors [1][2]

Memory Level	Access Pattern	Power Characteristics	Bandwidth Capability	Design Trade-off
Register File	Direct processor access	Minimal energy per operation	Limited by datapath width	Speed versus capacity
L1 Cache	Single-cycle hit latency	Low energy consumption	High throughput per core	Size versus associativity
L2 Cache	Multi-cycle access	Moderate energy cost	Moderate aggregate bandwidth	Capacity versus latency
L3 Cache	Longer access latency	Higher energy per access	Shared bandwidth resource	Inclusivity versus exclusivity
External DRAM	High latency penalty	Significant energy cost	Maximum system bandwidth	Capacity versus power

2. Cache Memory Architecture and Optimization for AI Workloads

Cache memory systems constitute the first line of defense against the memory wall problem, wherein processor performance increasingly exceeds memory system bandwidth. Modern ARM Cortex-A78 processors and similar high-performance cores implement sophisticated multi-level cache hierarchies that can dramatically reduce effective memory access latency when properly utilized. The Eyeriss architecture demonstrates that exploiting data reuse through hierarchical memory organization can reduce energy consumption dramatically, with register file accesses consuming 0.05 picojoules, scratchpad memory (SRAM) consuming 0.7 picojoules per access, and external DRAM requiring 200 picojoules per access, establishing a 4000× energy ratio between registers and DRAM [3]. However, the access patterns inherent to neural network inference present unique challenges for traditional cache architectures designed around temporal and spatial locality assumptions derived from general-purpose computing workloads. Analysis of

AlexNet's convolutional layers reveals three primary forms of data reuse: convolutional reuse where the same filter weight is applied across different spatial positions, filter reuse where multiple filters process the same input feature map position, and image reuse where the same input activation is used by multiple filters, collectively providing opportunities for 10× to 1000× reduction in memory bandwidth when properly exploited [3].

The hierarchical organization of L1, L2, and L3 caches reflects fundamental trade-offs between access latency, storage capacity, and power consumption. L1 caches typically provide single-cycle access to 32-64KB of data with a simple direct-mapped or 2-way set-associative organization. L2 caches extend capacity to several hundred kilobytes with 4-8 cycle latency and higher associativity, while L3 caches may reach tens of megabytes with commensurately longer access times. For AI inference workloads, the effectiveness of this hierarchy depends critically on whether weight tensors, activation maps, and intermediate results exhibit sufficient reuse to justify cache occupancy. The Eyeriss spatial architecture implements a 168-processing-element array with local 108 KB of register files and a 1 MB global buffer, achieving energy efficiency of 1.43 TOPS/W at 200 MHz by maximizing data reuse within the on-chip memory hierarchy and minimizing costly DRAM accesses [3]. Experimental evaluation demonstrates that processing AlexNet achieves 116 fps throughput while consuming only 278 mW, with the memory hierarchy design contributing 65% of the total energy savings compared to traditional architectures lacking hierarchical data reuse optimization [3].

Cache coherency protocols present particular challenges in multi-core inference scenarios. When multiple CPU cores collaborate on neural network inference, the coherency mechanism must track and propagate modifications to shared data structures. Research on virtualized deep neural networks reveals that GPU memory capacity limitations create significant bottlenecks, with memory consumption reaching 3.1 GB for VGG-19 and 4.7 GB for ResNet-152, exceeding the capacity of many embedded GPUs [4]. The vDNN system demonstrates that intelligently managing memory hierarchy through layer-wise memory virtualization can reduce peak memory footprint by 83% while maintaining 95% of baseline performance, enabling execution of networks that would otherwise fail due to memory constraints [4]. Quantitative analysis shows that careful partitioning of inference tasks to minimize inter-core communication can reduce cache coherency traffic by 60-80%, directly translating to improved throughput and reduced power consumption.

Table 2: Data Reuse Strategies for Convolutional Neural Networks [3][4]

Reuse Type	Memory Hierarchy Level	Optimization Target	Performance Benefit	Implementation Approach
Convolutional Reuse	Local scratchpad	Filter weight locality	Reduced DRAM bandwidth	Spatial data mapping
Filter Reuse	Shared buffer	Activation sharing	Enhanced throughput	Multi-filter parallelism
Image Reuse	Global memory	Input feature maps	Energy efficiency	Layer fusion techniques
Layer-wise Virtualization	System DRAM	Peak memory footprint	Capacity expansion	Dynamic memory swapping

3. Tightly Coupled Memory and Deterministic Access Patterns

Tightly coupled memory (TCM) represents a fundamentally different approach to the memory hierarchy problem, prioritizing determinism over statistical optimization. Unlike caches, which rely on dynamic replacement policies and probabilistic hit rates, TCM provides guaranteed single-cycle access to explicitly

managed memory regions. Research on scaling binarized neural networks demonstrates that utilizing on-chip Block RAM achieves deterministic performance, with implementations reaching 12,274 million operations per second per DSP on Xilinx Zynq platforms when weights remain entirely in FPGA fabric memory [5]. This architectural feature proves indispensable for real-time AI applications in safety-critical domains such as automotive perception systems, industrial automation, and medical device control, where the elimination of cache-induced timing variability ensures compliance with hard real-time constraints.

The architectural integration of TCM varies across processor implementations, but the defining characteristic remains a direct connection to the processor pipeline without intervening cache logic. ARM Cortex-R and Cortex-M processors commonly incorporate instruction TCM (ITCM) and data TCM (DTCM) ranging from tens to hundreds of kilobytes. The ARM Cortex-M7 processor features ITCM and DTCM interfaces that support configurations from 0 KB up to 16 MB per interface, with both memories delivering zero wait state access at the full processor clock frequency [6]. The TCM interfaces connect via dedicated 64-bit AHB-Lite buses directly to the processor core, operating independently of the optional 4-way set-associative data cache and instruction cache that can range from 4 KB to 64 KB each [6]. The programmer exercises complete control over TCM contents through linker script directives and explicit memory mapping, eliminating the non-determinism inherent to cache replacement policies. For neural network inference, this enables placement of time-critical code paths and frequently accessed weight matrices in TCM, guaranteeing bounded worst-case execution time.

The utilization of TCM for AI workloads requires careful analysis of memory access patterns and inference pipeline structure. A representative approach involves partitioning the neural network into stages, loading weights for each stage into TCM during initialization, and processing input tensors through successive stages with predictable timing. For modest networks—such as MobileNetV2 variants optimized for microcontroller deployment—entire model weights may reside in TCM, eliminating external memory accesses during the inference critical path. Experimental evaluation of binarized ResNet architectures demonstrates that complete on-chip weight storage enables inference rates of 12,053 frames per second for CIFAR-10 classification when operating at 200 MHz, with the deterministic memory access pattern ensuring zero variance in frame processing time [5]. Larger networks necessitate explicit management of TCM contents, swapping weight matrices as the inference pipeline progresses through network layers.

Integration with real-time operating systems (RTOS) introduces additional considerations for TCM utilization. The deterministic nature of TCM access makes it attractive for storing RTOS kernel code and interrupt service routines, potentially competing with AI workload requirements. Memory protection units (MPU) can partition TCM into regions with distinct access permissions, enabling secure coexistence of safety-critical control code and AI inference tasks. The Cortex-M7 implements an optional MPU supporting up to 16 independently programmable memory regions with configurable access permissions, size attributes, and memory type specifications [6]. Empirical measurements demonstrate that careful TCM allocation can reduce inference latency variance by 95% compared to cache-based implementations, crucial for meeting ISO 26262 timing requirements in automotive applications.

Table 3: Deterministic Memory Architectures for Real-Time AI [5][6]

Architecture Component	Timing Guarantee	Control Mechanism	Capacity Range	Application Domain
Instruction TCM	Zero wait state	Explicit address mapping	Kilobytes to megabytes	Safety-critical code
Data TCM	Deterministic access	Linker script allocation	Kilobytes to megabytes	Time-critical tensors
FPGA Block RAM	Guaranteed latency	Hardware synthesis	Megabits on-chip	Complete small models

Memory Protection Unit	Access isolation	Region-based permissions	Address space partitioning	Multi-domain systems

4. External Memory Controller Optimization and Bandwidth Management

External memory controllers bridge the performance gap between on-chip memory resources and the gigabytes of DRAM required for contemporary AI models. Modern embedded SoCs employ sophisticated DDR4, DDR5, or LPDDR memory controllers capable of sustaining hundreds of gigabits per second of aggregate bandwidth. Research on the DianNao neural network accelerator demonstrates that a 3.02 mm² silicon implementation fabricated in 65nm CMOS technology achieves 452 billion operations per second while consuming 485 mW, but this performance critically depends on memory subsystem optimization to feed the computational units [7]. However, the achievable bandwidth depends critically on access pattern characteristics, controller configuration, and quality-of-service (QoS) arbitration policies that mediate competing demands from multiple bus masters.

The physical characteristics of DDR memory impose fundamental constraints on access efficiency. DRAM organization into banks, rows, and columns necessitates multi-cycle command sequences for read and write operations. Opening a row incurs tens of nanoseconds of latency, while subsequent accesses to the same row benefit from reduced latency. Consequently, access patterns exhibiting strong row locality achieve significantly higher effective bandwidth than patterns requiring frequent row changes. Neural network inference workloads often demonstrate poor row locality due to strided access patterns in convolutional operations and scattered access patterns in attention mechanisms, necessitating careful data layout optimization. The DianNao architecture addresses this through a three-tier memory hierarchy: a 32 KB neuron buffer (NBin) for input neurons, a 64 KB neuron buffer (NBout) for output neurons, and a 256 KB synapse buffer (SB) for network weights, achieving energy efficiency of 0.9 pJ per operation for 16-bit fixed-point arithmetic compared to 640 pJ for external DRAM access [7]. This 711× energy advantage for on-chip storage motivates aggressive data reuse strategies, with the architecture achieving 188× better energy efficiency than GPUs through localized memory access patterns [7].

Between rival agents, including CPU clusters, GPU cores, artificial intelligence accelerators, and I/O systems, the memory controller QoS mechanism regulates bandwidth distribution. Priority-based arbitration strategies guarantee that static or dynamic priorities assigned to transaction streams guarantee that latency-sensitive control traffic stops best-effort data transfers. Bandwidth reservation mechanisms guarantee minimum bandwidth allocation to specific agents regardless of system load. The ARM Mali-G710 GPU implements sophisticated memory management capabilities with a 2 MB L2 cache partitioned across four slices, providing aggregate bandwidth sufficient to support up to 16 shader cores operating at frequencies up to 1 GHz [8]. For AI inference pipelines that interleave data acquisition, preprocessing, neural network evaluation, and post-processing, proper QoS configuration prevents starvation scenarios where inference engines await memory access while lower-priority transactions consume available bandwidth.

Advanced memory controller features, including bank-level parallelism, command reordering, and adaptive page policies, significantly impact AI inference performance. Modern controllers can maintain multiple open pages across different banks, enabling concurrent servicing of requests to distinct memory regions. Out-of-order execution windows allow controllers to resequence memory commands to maximize row hits and minimize idle cycles. The Mali-G710 incorporates transaction elimination technology that reduces memory bandwidth consumption by up to 30% through intelligent tile-based rendering and frame buffer compression, effectively multiplying available external memory bandwidth [8]. Quantitative analysis of production automotive systems reveals that expert configuration of these parameters can improve inference throughput by 40-60% compared to default settings, highlighting the importance of memory subsystem tuning in overall system optimization.

Table 4: Memory Controller Optimization Features [7][8]

Controller Feature	Efficiency Mechanism	Configuration Method	Workload Suitability	System Impact
Multi-bank Operation	Parallel request servicing	Hardware scheduling	Distributed access patterns	Throughput enhancement
Command Scheduling	Request reordering	Priority policies	Variable locality workloads	Latency reduction
Page Management	Open versus closed policy	Adaptive algorithms	Pattern-dependent loads	Efficiency optimization
Transaction Compression	Data reduction techniques	Transparent operation	Redundant content streams	Effective bandwidth increase

5. AI-Specific Memory Optimization Techniques

The distinctive characteristics of neural network inference workloads motivate the development of specialized memory optimization strategies beyond traditional embedded systems techniques. Model quantization emerges as a particularly effective approach, reducing both memory capacity requirements and bandwidth consumption by representing weights and activations with reduced numerical precision. Research on quantization techniques demonstrates that INT8 quantization of ResNet-50 achieves 76.4% top-1 accuracy on ImageNet, representing only a 0.5% degradation compared to the FP32 baseline accuracy of 76.9%, while reducing model size from 102 MB to 25.5 MB [9]. The transition from 32-bit floating-point representation to 8-bit integer format reduces memory footprint by 75% while maintaining acceptable accuracy for many perception tasks. Advanced quantization schemes incorporating per-channel scaling factors and mixed-precision operations enable deployment of billion-parameter language models on embedded systems with as little as 8GB of DRAM. Experimental analysis reveals that asymmetric quantization with per-channel granularity for weights and per-tensor granularity for activations provides optimal accuracy-efficiency trade-offs, with MobileNet-v2 achieving 71.8% top-1 accuracy using INT8 compared to 72.0% with FP32 while reducing memory bandwidth requirements by 4× [9].

The implementation of quantization optimization spans multiple levels of the software stack. Deep learning frameworks such as TensorRT and ONNX Runtime provide automated quantization workflows that analyze activation statistics during calibration, determine optimal scaling factors, and generate optimized inference engines. Hardware support for integer arithmetic operations, including dot-product instructions and vector processing extensions, ensures that quantized models execute efficiently on embedded processors. The NVIDIA Jetson Xavier NX platform, featuring 384 CUDA cores, 48 Tensor cores, and 6 Carmel ARM CPU cores, delivers up to 21 TOPS of INT8 inference performance compared to 6 TFLOPS of FP16 performance, demonstrating a 3.5× computational advantage for quantized models [10]. The memory bandwidth reduction resulting from quantization proves particularly beneficial for memory-bound inference workloads, where computation proceeds faster than data can be supplied from external memory. The Xavier NX implements a 128-bit memory interface supporting LPDDR4x memory at 51.2 GB/s bandwidth, and quantization-enabled models effectively multiply this bandwidth by utilizing 4× fewer bits per parameter [10].

Memory pool management strategies address the dynamic memory allocation patterns inherent to neural network inference. Traditional heap allocators, designed for general-purpose workloads with unpredictable allocation patterns, introduce fragmentation and non-deterministic latency unsuitable for real-time AI systems. Custom allocators designed for inference workloads exploit knowledge of tensor sizes and lifetime to implement efficient memory recycling. Pre-allocated pools sized to accommodate maximum tensor dimensions eliminate runtime allocation overhead, while arena allocators satisfy all allocations from contiguous memory regions that can be freed wholesale at inference completion. The Xavier NX features 8 GB of LPDDR4x memory shared between CPU and GPU, necessitating careful memory management to support concurrent workloads [10].

Zero-copy data flow optimization minimizes memory traffic by eliminating intermediate buffer copies between inference pipeline stages. This approach requires careful coordination of memory allocation such that output tensors from one processing stage occupy memory regions directly accessible by subsequent stages. DMA-coherent memory regions facilitate data transfer between the CPU and specialized accelerators without explicit cache maintenance operations. The Xavier NX supports unified memory architecture, enabling zero-copy access between CPU and GPU, with the NVDLA deep learning accelerator capable of direct memory access to shared DRAM without intermediate CPU intervention [10].

Conclusion

The memory hierarchy for embedded AI systems is an intricate interplay between architectural design decisions, workload properties, and optimization techniques collectively defining expected performance, power efficiency, and economic viability. The evolution from smaller yet faster cache memory to tightly coupled memory to outside DRAM controllers mirrors basic capacity-latency-determinism-cost trade-offs. Effective deployment of AI workloads on low-resource embedded platforms demands a deep understanding of these trade-offs and their system design implications. Empirical evidence shows that systematic optimization of the memory subsystem leads to substantial performance improvements, including increased inference throughput, lower memory subsystem power consumption, and enabling significantly larger models on cost-constrained hardware. These quantitative improvements are achieved by diverse optimization strategies, including cache-aware data layout, deterministic tightly coupled memory usage, bandwidth management through quality of service configuration, model quantization, and special memory allocation schemes. Combining these technologies allows for the application of complex artificial intelligence features on systems from high-performance vehicle computers to microcontroller-based edge devices. Embedded artificial intelligence systems will keep evolving, therefore exacerbating memory hierarchy problems as model complexity grows and real-time performance standards tighten. Promising to revolutionize the optimization terrain are new memory technologies, including non-volatile memory integration, processing-in-memory architectures, and high-bandwidth memory. Nevertheless, fundamental principles underlying memory hierarchy design, such as the interplay between capacity and access latency, the bandwidth vs. power trade-off, and the conflict between statistical optimality and determinism, will remain to direct embedded systems engineers in their design of efficient AI solutions. Dominance of these principles, together with sound performance analysis and systematic optimization techniques, forms a fundamental competence for engineers operating at the boundary of embedded systems and artificial intelligence.

References

[1] Vivienne Sze, et al., "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," IEEE, 2017. Available: https://ieeexplore.ieee.org/document/8114708

[2]Israel Zamudio, “ARM Cortex-A78 Core Technical Reference Manual," ARM Technical Reference Manual, Revision r1p1, 2018. Available: https://www.scribd.com/document/574792892/arm-cortex-a78-trm-101430-0101-05-en

[3] Yu-Hsin Chen, et al., "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," IEEE, 2016. Available: https://ieeexplore.ieee.org/document/7738524

[4] Minsoo Rhu, et al., "vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design," arXiv, 2016. Available: https://arxiv.org/abs/1602.08124

[5] Nicholas J. Fraser, et al., "Scaling Binarized Neural Networks on Reconfigurable Logic," ACM digital library, 2017, Available: https://dl.acm.org/doi/10.1145/3029580.3029586

[6] ARM Ltd., "Arm Cortex-M7 Processor Technical Reference Manual," 2018. [Online]. Available: https://developer.arm.com/documentation/ddi0489/latest/

[7] Tianshi Chen, "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," ACM Digital Library, Available: https://dl.acm.org/doi/10.1145/2654822.2541967

[8] ARM Ltd., "Arm® Mali™-G78 Performance Counters Reference Guide 1.9," 2021. [Online]. Available: https://developer.arm.com/documentation/102626/latest/

[9] Raghuraman Krishnamoorthi, "Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper," arXiv, 2018. [Online]. Available: https://arxiv.org/abs/1806.08342
[10] NVIDIA Developer, "Jetson Xavier NX Developer Kit - Get Started,". [Online]. Available: https://developer.nvidia.com/embedded/learn/get-started-jetson-xavier-nx-devkit