A Coherence-Aware Runtime Reconfigurable System on Chip for Efficient Language Model Inference

Je Yang, *Columbia University, New York, NY, 10027, USA*
Gracen Wallace, *IBM Research, Yorktown Heights, NY, 10598, USA*
Joseph Zuckerman, Gabriele Tombesi, and Luca P. Carloni, *Columbia University, New York, NY, 10027, USA*

Generative artificial intelligence (AI), powered by Transformer-based language models (LMs), such as OpenAI's ChatGPT, is rapidly generating a new digital landscape. Unfortunately, it is hard to get efficient LM inference by simply scaling up the compute elements, particularly on resource-constrained edge AI platforms. In this context, we revisit a critical yet often overlooked system component: the memory subsystem. We propose a reconfigurable system-on-chip (SoC) architecture that dynamically modulates the cache-coherence modes to align with the heterogeneous demands of generative LM workloads. Our coherence-aware SoC outperforms a commercial edge GPU by yielding 8.12× higher system efficiency when running the OPT-6.7B model, highlighting the potential of system-level adaptability as a key enabler for efficient and scalable LM inference.

Generative artificial intelligence (AI) has rapidly emerged as an unprecedented force in computing, driven by the success of Transformer-based language models (LMs) that generate textual and visual content with ground-breaking sophistication.[1,2,3] This technological shift has accelerated commercial adoption, with the LM market valued at US$4.1 billion in 2023 projected to grow more than tenfold to US$53.9 billion by 2032. Such rapid growth amplifies the demand for efficient inference infrastructures for a rich user experience, especially on edge and embedded platforms with tight resource constraints.

However, delivering high-quality inferences comes at a substantial computational cost. The attention mechanism, a cornerstone of modern LMs, suffers from irregular data movement, low arithmetic intensity, and quadratic scaling with sequence length. For example, attention layers can take more than 40 s to execute on a resource-constrained ARM CPU, making interactive applications infeasible. These limitations worsen during token-by-token generation, which offers little opportunity for parallelism and requires repeated access to the model parameter with minimal reuse, making memory bandwidth and latency the primary bottlenecks.

Moreover, Transformers consist of computational kernels that are intrinsically heterogeneous, including dense vector-matrix multiplications, memory-bound element-wise operations, and bandwidth-intensive cross-attention. The performance and resource demands of each kernel vary significantly with the model size and the length of the input–output (I/O) sequence. For example, as the input length increases, the memory activity grows more than three times.[4] This diversity makes fixed-function accelerators or a single hardware configuration impractical.

Prior efforts have mainly focused on reducing computational complexity by scaling accelerators for large-scale models, while substantially overlooking a critical source of system inefficiency: *the memory subsystem*.

Digital Object Identifier 10.1109/MM.2026.3689025
Date of publication 1 May 2026; date of current version 17 June 2026.

In particular, cache-hierarchy configurations and coherence modes remain largely underexplored for LM inference. This oversight leads to inefficiencies particularly during the generation phase, where matrix transposes occur at every token step, repeatedly accessing the same memory regions. As a result, coherence-aware data sharing can have a substantial effect on performance.

To address this gap, we adopt a system-level perspective that reexamines how existing compute and memory resources are orchestrated instead of redesigning accelerators or merely scaling hardware for each new model. We propose a *coherence-aware reconfigurable system-on-chip (SoC) architecture* that dynamically adjusts the inter-accelerator parallelism level and selects appropriate cache-coherence modes on a per-layer basis. By supporting layer-wise invocation, a runtime software stack allows the SoC to adapt to the computational characteristics of each Transformer block. By decoupling hardware specialization from software configuration, our approach supports model-agnostic deployment and accelerates time-to-market.

Our comprehensive evaluation demonstrates that runtime adaptability unlocks substantial gains in system performance by navigating the Pareto frontier between coherence-induced latency and resource utilization. Relative to a static noncoherent baseline, our adaptive strategy delivers a 1.17× speedup with 3.13× traffic reduction for BERT and a 1.12× speedup with 1.74× traffic reduction for the OPT family. Consequently, our coherence-aware configuration leads to an 8.12× higher system efficiency when running OPT-6.7B, compared to a commercial edge GPU. These findings underscore the importance of dynamic optimization of the memory system while balancing the limited resources for LM acceleration and suggest that system-level configurability may be as critical as model- or hardware-level optimization for the next generation of edge AI systems.

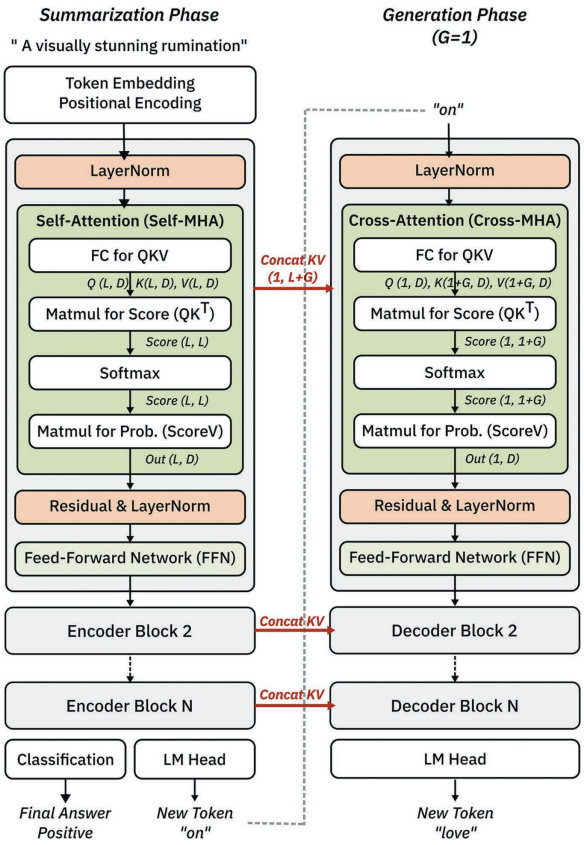


FIGURE 1. Language model architecture and its characteristics. *L* is the input sequence length, *D* the model dimension, *N* the number of encoder/decoder blocks, and *G* the generated sequence length.

BACKGROUND AND MOTIVATION

Structure of Language Models

Language models (LMs) operate in two phases: summarization and generation. As shown in Figure 1, these correspond to the encoder and decoder blocks of the Transformer architecture.[1] In summarization, the encoder embeds tokens into vectors and processes them through blocks containing a *multihead attention (MHA)* module and a *feed-forward network (FFN)*, each followed by layer normalization and a residual connection. Each encoder block projects its input to query (*Q*), key (*K*), and value (*V*) matrices, which are used to compute attention scores via head-wise matrix multiplications and softmax normalization. Since *Q*, *K*, and *V* come from the same input, this is called *self-attention*. The FFN consists of two fully connected layers with expanded dimensions and a *Gaussian error linear unit (GELU)* activation.

The generation phase exhibits two key differences. First, unlike the encoder, the decoder operates one token at a time, rather than processing the entire sequence in parallel. Second, the decoder uses the encoder's precomputed key and value matrices and concatenates them with its own representations, while the query remains a single vector corresponding to the current token. Because the query attends to external *K* and *V* rather than its own, this process is termed *cross-attention*. To generate *G* output tokens, the decoder must be invoked *G* times, each with an incrementally longer target sequence. This makes the generation phase inherently sequential and

limits opportunities for parallelism, whereas the summarization phase benefits significantly from parallel computation.

Challenges in LM Acceleration

The sequential nature of LM inference demands continuous access to new parameters with limited reuse, creating a pronounced memory bottleneck. Scaling compute or memory bandwidth often fails to yield proportional performance gains, especially during autoregressive generation. Although modern GPUs offer high computational throughput and memory bandwidth, they are optimized for massively parallel workloads. When processing a single token, these resources cannot be efficiently directed to the tensor cores that handle vector-level operations, thereby leaving both compute and memory underutilized.

Beyond the challenges of parallelism, Transformer models also exhibit substantial computational heterogeneity across kernel types and sequence lengths. Figure 2 shows the profiled floating-point operations (FLOPs) and memory operations (MOPs) for the encoder and decoder blocks when running two widely adopted Transformer-based LMs on a single NVIDIA GPU: BERT[2] for encoder-based summarization tasks and OPT[3] for decoder-based generation.

THIS VARIATION HIGHLIGHTS THE NEED FOR FINE-GRAINED ARCHITECTURAL ADAPTABILITY THAT GOES BEYOND ONE-SIZE-FITS-ALL HARDWARE ACCELERATION.

For short sequences, vector-matrix operations in the MHA and FFN layers dominate the FLOPs. However, as the sequence length increases, matrix–matrix multiplications become the primary compute driver, whereas memory-intensive element-wise operations emerge as the dominant contributor to MOPs in the encoder. The decoder shows similar trends, though with consistently lower sensitivity to element-wise operations. This variation highlights the need for

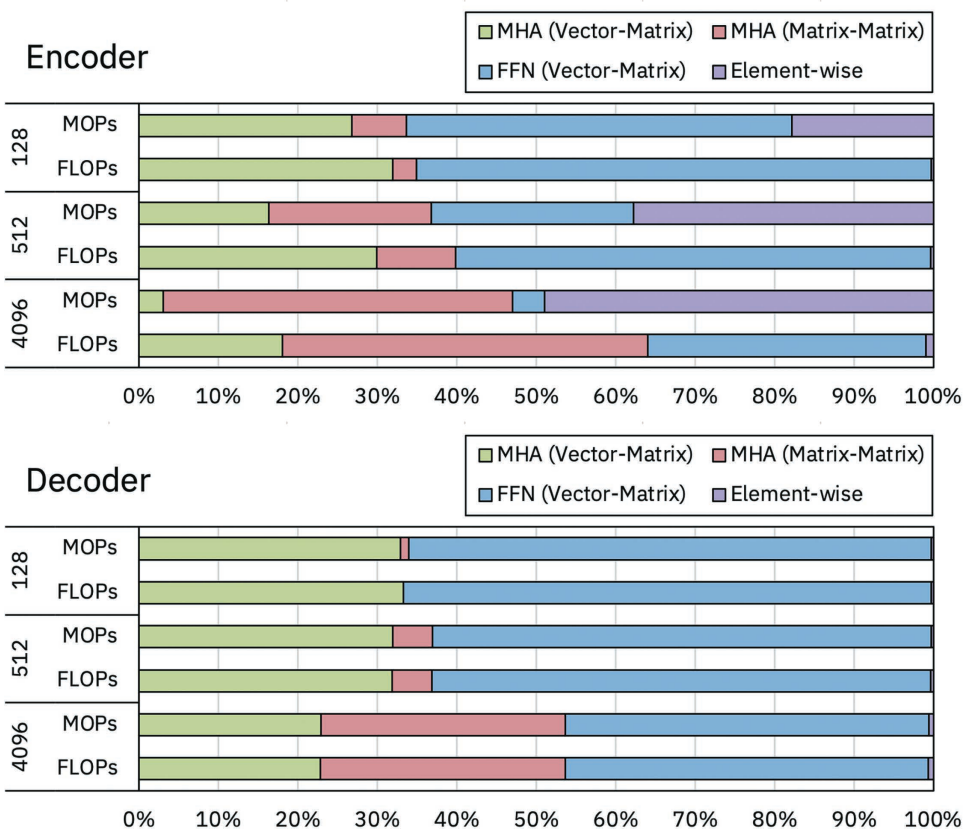


FIGURE 2. Per-layer FLOPs and MOPs for each encoder and decoder running BERT and OPT, respectively with sequence lengths of 128, 512, and 4096. MOP is the total number of load and store operations.

fine-grained architectural adaptability that goes beyond one-size-fits-all hardware acceleration.

In generative workloads, decoder layers repeatedly access shared key–value matrices with minimal reuse, making coherence-aware data sharing highly beneficial. Yet, existing systems lack the flexibility to exploit it. These findings expose a fundamental challenge: *Heterogeneous kernel types impose conflicting demands on the memory hierarchy, making any single static coherence configuration suboptimal across Transformer layers.* To address this, we present a cache-coherence-aware, runtime-reconfigurable SoC that shifts from raw scaling to intelligent orchestration via per-layer coherence adaptation driven by each kernel's arithmetic intensity and reuse pattern. Our analysis further reveals two nonobvious cache pathologies—the saturation token length and the capacity cliff—that together motivate and validate dynamic reconfigurability across model scales and generation lengths.

LM ACCELERATOR DESIGN

Accelerator Microarchitecture

We designed the LM accelerator, the main component of our SoC, with a system-level design approach that leverages latency-insensitive design and high-level synthesis to enable faster simulation and broader design-space exploration.[5] As shown in Figure 3, the microarchitecture of our LM accelerator consists of four main modules: *instruction controller, data arrange unit, reusable scratchpad* and *vector execution engine*.

The instruction controller orchestrates the overall execution by decoding configuration registers from the host into control signals for the other units. It arranges the complex dataflow sequences required for operation fusion, managing dependencies between computational stages and ensuring precise synchronization across the parallel processing in the vector execution engine and memory interfaces.

The data arrange unit interfaces with the external memory subsystem via dedicated direct memory access (DMA) channels. Its primary function is the efficient preprocessing and organization of data streams, meticulously arranging incoming data into formats suitable for the vector execution engine and packing outgoing results. A key capability is its on-the-fly transformation logic, which, for instance, transposes the key matrix as it is streamed from memory during MHA. This hardware-level data manipulation eliminates explicit software-managed data reshaping or costly round-trips to off-chip memory, significantly reducing both latency and bandwidth consumption.

The reusable scratchpad (RS) consists of multiple multibanked on-chip SRAMs, with the number of banks matched to the vector lane size (V) to maintain high-throughput streaming. To mitigate limited on-chip capacity, we adopt aggressive in-place memory reuse. In self- and cross-MHA, operations are fused to minimize intermediate memory traffic, especially

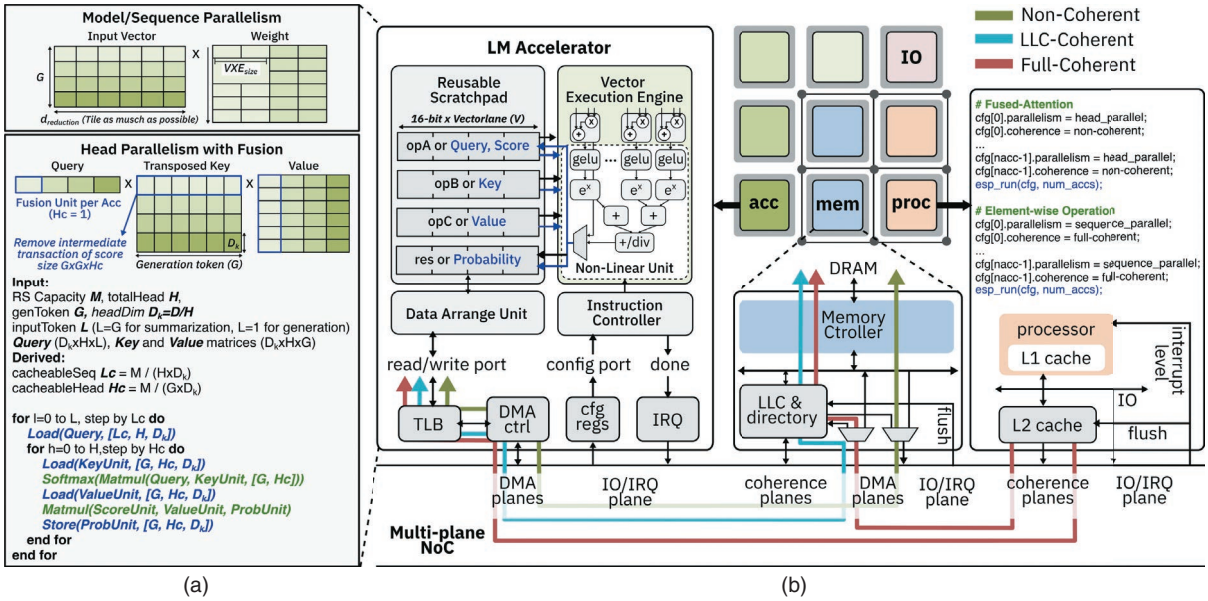


FIGURE 3. LM accelerator with coherence-aware runtime reconfigurable SoC. (a) Dataflow with hybrid parallelism. (b) Overview of SoC architecture with 3×3 tile grid with layer-wise invocation.

for attention scores and softmax-normalized matrices. During fused attention, the query, key, and value matrices are loaded into the *opA*, *opB*, and *opC* buffers, respectively. After *opA* (query) is consumed in the first matrix multiplication(query·key^T), its space is immediately reused to store the softmax-normalized scores. This reuse effectively halves the intermediate memory footprint, enabling longer sequence support under tight capacity constraints. For nonfused operations, the RS supports a uniform computation pattern, $res = opA \times opB + opC$, covering a broad range of linear and element-wise Transformer operations.

The vector execution engine (VXE) is the accelerator's main computing core, designed to execute all vector–matrix multiplication, matrix–matrix multiplication (in MHA and FFN), and element-wise operations (softmax, layer normalization, and residual addition). It contains V multiply-and-accumulate (MAC) units arranged in a 1-D vector, followed by a shared, highly pipelined nonlinear unit containing dedicated GeLU and exponential operators. By decomposing complex nonlinear operations into these fundamental components, the VXE exploits the common computational patterns found across modern LMs, enabling support for a wide range of architectures while minimizing hardware area overhead.

Accelerator Dataflow

Our LM accelerator employs an output-stationary dataflow: Input vectors are broadcast to all V MAC units, while weight matrices are streamed during vector–matrix multiplications. This configuration enables the concurrent computation of V output elements, maximizing output-level parallelism. To further enhance efficiency, our scheduler prioritizes localizing the entire accumulation dimension within a single tile, thereby eliminating the overhead of storing and retrieving intermediate partial sums from off-chip memory.

Building upon this stationary dataflow, we employ a hybrid intra-layer parallelism strategy to effectively manage the large scale of Transformer models. We apply model/sequence parallelism for non-MHA operations such as FFN layers while utilizing head parallelism with kernel fusion for the MHA operation. For FFN layers, weight matrices are partitioned column-wise or sequence-wise based on the dimension, ensuring each accelerator can efficiently process its assigned data partition independently.

For MHA in both the encoder and the decoder, we employ tightly coupled operation fusion while partitioning the data head-wise, with the Algorithm reported in Figure 3(a). The fusion process begins with a runtime calculation to determine the maximum number of query tokens (L_c) and attention heads (H_c) that can be accommodated within the available capacity of on-chip RS (M). Based on this constraint, a partial query matrix of shape $[L_c, H_c, D_k]$ is preloaded into the RS. In generation mode ($L = 1$), this preload step occurs only once. For each block of H_c attention heads, the corresponding transposed key and value matrices ($[G, H_c, D_k]$) are fetched in a single contiguous transfer, where G is the number of tokens in the output sentence. The accelerator then executes the fused kernel: First, it computes the attention scores via a matrix multiplication between the preloaded query matrix and the streamed key matrix. This is immediately followed by an on-the-fly row-wise softmax operation. The normalized scores are then directly consumed in a second matrix multiplication with the streamed value matrices to generate the final attention outputs of shape $[G, H_c, D_k]$, which are subsequently written back to main memory. This fused execution loop continues until all H attention heads are processed. A key advantage of this fused dataflow is that the intermediate attention score tensor is both generated and consumed entirely on-chip, eliminating the need for off-chip memory transfers and enhancing the compute-to-data locality. This benefit becomes increasingly pronounced as the context length G grows because it avoids the expensive external transfer of a large $G \times H$ score tensor, especially when $G \gg H$.

BUILDING UPON THIS STATIONARY DATAFLOW, WE EMPLOY A HYBRID INTRA-LAYER PARALLELISM STRATEGY TO EFFECTIVELY MANAGE THE LARGE SCALE OF TRANSFORMER MODELS.

SYSTEM DESIGN

SoC Architecture

Building on our LM accelerator, we designed a tile-based heterogeneous SoC to exploit coarse-grained parallelism by mapping kernel partitions to separate accelerator instances for higher throughput and scalability. To design our SoC, we leveraged ESP,[6] an open source SoC platform with a scalable tile-based architecture and flexible system-level design methodology. The system incorporates four primary tile types (processor tile, memory tile for communication with main

memory DRAM, accelerator tile, and auxiliary tile for I/O peripherals and system utilities) that communicate via a multiplane network-on-chip (NoC).

Processor and Memory Tiles

The processor tile serves as the system's orchestrator, dispatching tasks and configuration commands to each accelerator. It features a 64-bit RISC-V CPU core with private L1 and L2 caches. Each memory tile has a DDR controller and a configurable partition of the last-level cache (LLC).

THIS FLEXIBILITY ALLOWS THE SAME ACCELERATOR TO SWITCH CACHE-COHERENCE MODES AT RUNTIME DEPENDING ON THE CHARACTERISTICS OF THE WORKLOAD BEING PROCESSED.

Accelerator Tiles

In ESP, a specialized hardware accelerator is encapsulated within a standardized socket providing NoC interface, dedicated DMA, virtual-to-physical address translation with a lightweight translation look-aside buffer (TLB), memory-mapped configuration registers, and interrupt mechanisms to notify task completion. An accelerator can communicate with the main memory subsystem through the following three main cache-coherence modes, which are supported by the NoC[7]:

Noncoherent mode (Non-M): The accelerator accesses the main memory using DMA transactions over dedicated NoC planes, bypassing the entire cache hierarchy.

LLC-coherent mode (LLC-M): DMA requests from the accelerator are routed through the LLC, allowing reuse of cached data while bypassing the processors' private caches. This approach reduces off-chip traffic when data reside in the LLC, with DMA traffic flowing through coherence-enabled NoC planes.

Fully coherent mode (Full-M): Memory requests from the accelerator can be served at every level of the memory hierarchy. A coherence bridge ensures automatic invalidation of stale cache lines, eliminating software-managed cache flushing. This flexibility allows the same accelerator to switch cache-coherence modes at runtime depending on the characteristics of the workload being processed.

Layer-Wise Accelerator Invocation

A key contribution of our system is a layer-wise invocation model that exploits per-layer heterogeneity in Transformers. Beyond coarse-grained parallelism, it enables coarse-grained memory-hierarchy configurability by dynamically selecting the accelerator's cache-coherence mode for each layer, unlike traditional systems that statically assign hardware resources.

Each Transformer layer, whether a memory-bound MHA block or a compute-intensive FFN, is treated as an independent scheduling unit. At runtime, the system selects an appropriate coherence protocol. For example, matrix-multiplications in MHA modules may benefit from increased memory bandwidth and LLC sharing (via LLC-coherent mode), while FFN layers can leverage more compute tiles with relaxed coherence. Leveraging the ESP application programming interface,[6] the invocation process follows a three-step procedure:

1) Allocate shared memory buffers for inputs and outputs via `esp_alloc`.
2) Populate per-layer configuration registers in the target accelerator tile, including TLB entries, coherence settings, and tile groupings.
3) Trigger execution with `esp_run`, then release resources with `esp_free`.

This reconfiguration involves only register writes and TLB updates, incurring negligible software overhead and allowing rapid switching between heterogeneous layers without pipeline flushes or recompilation. By decoupling computation from fixed mappings and employing coherence-aware adaptability, our SoC supports model-agnostic deployment and scalable Transformer acceleration through intelligent runtime orchestration rather than hardware over-provisioning. Although prototyped on the ESP platform, the proposed coherence-aware runtime reconfiguration relies on standard cache-hierarchy interfaces that are broadly available in modern heterogeneous SoCs.

EVALUATION

Experimental Setup

Application Benchmarks

We evaluate our configurable SoC using a benchmark suite spanning two foundational LM paradigms: encoder-based BERT[2] and decoder-based OPT family.[3] We selected BERT-Base (110 M) for summarization tasks and a wide spectrum of OPT models, including OPT-350M, OPT-1.3B, and OPT-6.7B, for generative workloads. This diverse selection scaling up to a billion

parameters enables a robust analysis of cache-coherence behaviors across heterogeneous kernel characteristics. To maintain numerical consistency and efficiency, we employed 16-bit floating-point precision for all models.

System Implementation

We prototyped the SoC on a proFPGA UltraScale+ XCVU19P board with four accelerator tiles (V = 16), two memory tiles, a CPU tile, and an I/O tile. We allocated a 32-KB private L2 cache for each accelerator and processor tile, and a 0.25-MB LLC partition per memory tile. The choice of four accelerator tiles to two memory tiles is informed by prior design-space exploration,[8] which shows that adding more memory tiles yields diminishing performance returns relative to the associated resource overhead. Instead, sharing memory tiles across accelerators, together with coherence-mode selection to manage LLC contention, is more resource-efficient. To maximize on-chip density, our reusable scratchpad employs a hybrid BRAM-URAM architecture. Our design utilizes 18.7% of LUTs, 7.7% of registers, 65.0% of DSPs, 33.8% of BRAM, and 80.0% of URAM in total. Each accelerator dissipates only 1.1 W with 24.5% of the total DSPs, while the memory tiles consume 0.6 W, ensuring an energy-efficient integration.

Kernel-Level Performance

Figure 4(a) shows the tradeoff between coherence-driven performance and resource utilization across representative kernel classes. We report resource utilization as the fraction of total resources consumed by the instantiated accelerators, and we additionally include the cache resources attributable to the selected coherence mode (CPU L2 and LLC where applicable).

THIS DIVERSE SELECTION SCALING UP TO A BILLION PARAMETERS ENABLES A ROBUST ANALYSIS OF CACHE-COHERENCE BEHAVIORS ACROSS HETEROGENEOUS KERNEL CHARACTERISTICS.

For memory-intensive element-wise operations, dynamic adaptation of the cache-coherence mode proves critical. The Full-M achieves a 2.89× speedup across configurations, whereas the Non-M remains near 1× even with increased accelerator counts. This result demonstrates that for kernels with low arithmetic intensity, optimizing data movement via coherence protocols is more energy efficient and effective than raw hardware scaling. Conversely,

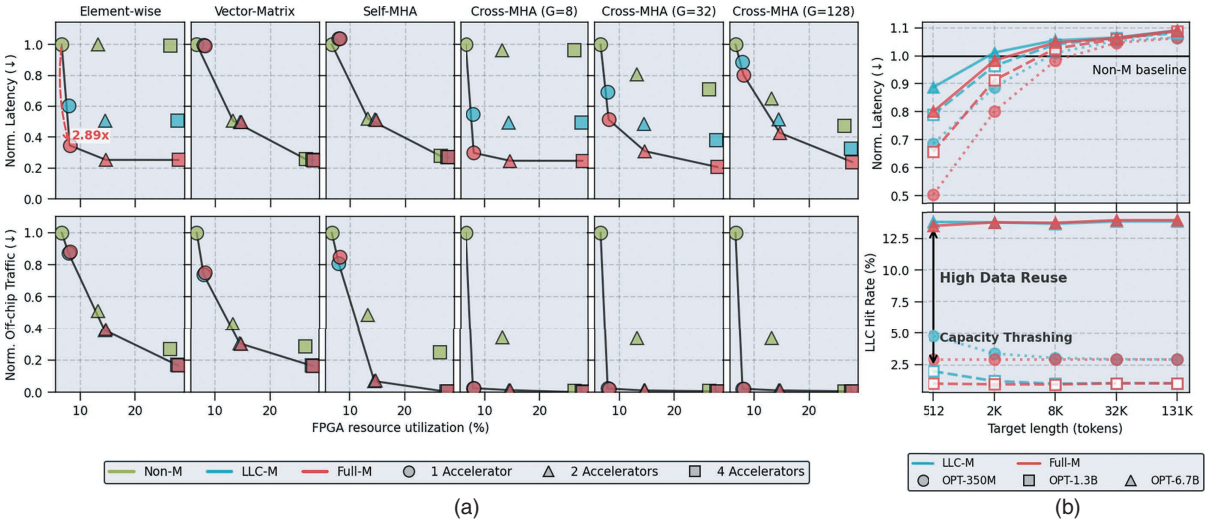
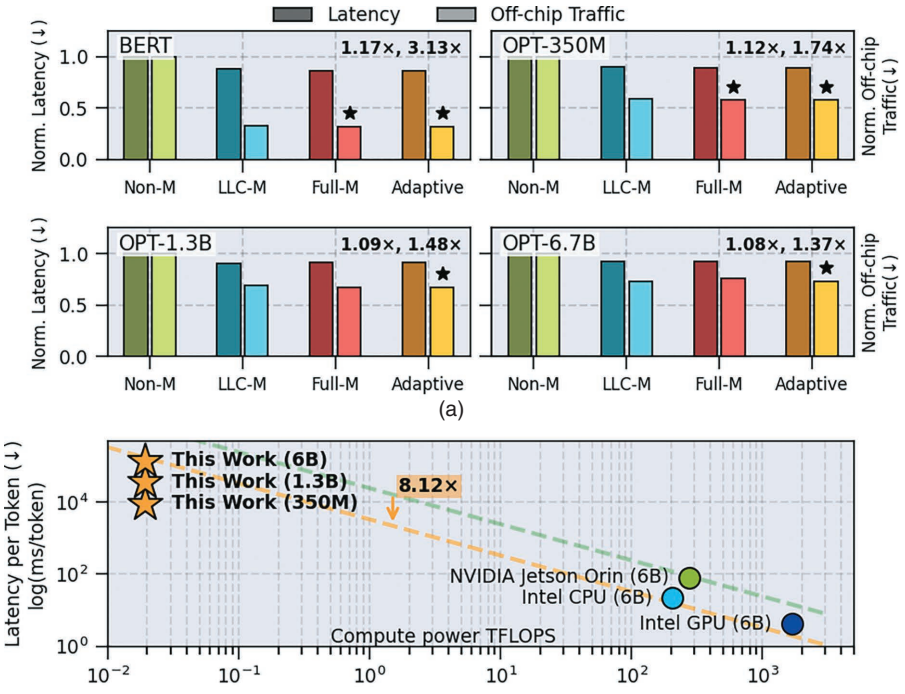


FIGURE 4. System performance analysis under varying memory optimization strategies. (a) Kernel-wise performance: A fixed input sequence length (L = 128) and the OPT-350M model configuration are used. The black line denotes the Pareto frontier points. (b) Scalability with generation length: Performance of the Cross-MHA kernel as the target generation token length (G) scales; the four-accelerator configuration is used and the latency is normalized to the noncoherent mode.

compute-bound vector-matrix multiplication shows limited sensitivity to coherence modes: Performance gains here are primarily driven by the scalability of accelerator tiles through our hybrid model/sequence parallelism.

For attention mechanisms, operator fusion reduces off-chip access for intermediate scores by up to 2.87×. However, coherence benefits vary by kernel type. For self-MHA, which is parallelizable across the input sequence, the Non-M yields optimal performance under resource-constrained scenarios (e.g., single accelerator) due to the frequent cache flushing for large footprints of query, key, and value tensors. Cross-MHA, by contrast, benefits significantly from coherence owing to its memory-bound nature. While LLC-M delivers progressively larger reductions in off-chip memory accesses as model scale increases, reaching up to 47.6× for OPT-6.7B, its latency benefit diminishes as the key–value (KV) matrix grows with the longer generation token lengths (G). This reveals that the performance bottleneck shifts from off-chip bandwidth to coherence protocol overhead as the volume of coherent requests scales with G.

Figure 4(b) further analyzes this sensitivity to generation length (*G*) and model size. Unlike standard CPU policies that offload KV data to DRAM,[9] our results reveal that maintaining KV cache in the LLC/private cache provides significant latency reduction until the saturation point is reached for edge-deployment. At short contexts (G = 512), Full-M execution yields up to 1.98× speedup relative to the Non-M baseline, as the KV working set still fits reasonably well within the private caches and the LLC. However, as *G* increases, the total KV footprint grows



Specification	Edge GPU (Jetson Orin)	Server GPU (Intel Max 1550)	Server CPU (Xeon Max)	This Work
Frequency	1.2 GHz	0.9 GHz	2.1 GHz	0.1 GHz
Compute Unit	2048 CUDA Cores 64 Tensor Cores	128 Xe-Cores 1024 XMX Engines	48 Cores (AMX Supported)	96 MACs
On-Chip Memory	6 MB L2	408 MB L2 64 MB L1	105 MB L3 96 MB L2	2MB RS 0.5MB LLC
Off-Chip BW	205 GB/s	3277 GB/s	1280 GB/s	22.5 GB/s

(b)

FIGURE 5. End-to-end system performance and comparison with commercial platforms. (a) The black star * denotes the coherence mode with minimum off-chip traffic. (b) Yellow and green diagonal lines represent iso-efficiency contours derived from latency per token × compute power of our reconfigurable SoC and the NVIDIA Jetson GPU, respectively.

linearly, eventually exceeding the *saturation token length*—the inflection point where streaming accesses bypass reuse, causing coherent latency to degrade below the Non-M baseline. Empirical results confirm that this saturation point is reached earlier for the larger OPT-6.7B compared to OPT-350M because of its massive parameter density.

Most notably, we observe a nonmonotonic relationship between model size and cache efficiency that we define as *capacity cliff*. The OPT-1.3B model specifically suffers from severe capacity thrashing and exhibits lower hit rates than the smaller OPT-350M model because its increased number of heads (H) extends the data reuse interval of the attention loop beyond the effective capacity of the cache hierarchy, causing valid lines to be evicted before reuse. In a counter-intuitive reversal of this trend, the significantly larger OPT-6.7B model recovers a much higher LLC hit rate of approximately 13.8%. This restoration is driven by the interaction between our fused row-wise softmax tiling strategy and the model's increased dimension per each head (D_k). Unlike the 1.3B model ($D_k = 64$), the larger head dimension of the 6.7B model improves spatial locality by saturating cache lines more effectively during contiguous memory accesses. Our fusion kernel exploits this alignment to maximize data reuse within each fetched cache line before eviction. Consequently, the system overcomes the raw capacity penalty of the larger model by achieving superior cache line utilization. This behavior validates the necessity for dynamic reconfiguration to navigate the tradeoff between capacity limits and reuse benefits, ensuring robust generalizability across the full spectrum of model scales.

System-Level Performance

Building on our kernel-level analysis, we evaluated the holistic end-to-end inference performance, comparing fixed-coherence modes with our proposed adaptive strategy. Guided by a Pareto frontier analysis, we configure the adaptive system with the maximum number of accelerators and a fully enabled cache hierarchy, allowing the scheduler to dynamically select the optimal coherence mode for each kernel. As illustrated in Figure 5(a), our adaptive approach consistently delivers superior efficiency by achieving a 1.17× speedup and substantial 3.13× reduction in off-chip traffic for encoder-dominant model BERT, while achieving a 1.12× speedup and 1.74× reduction in bandwidth consumption for the decoder-based OPT family, compared to baseline Non-M. Notably, the benefits of the adaptive policy become distinct as model scale grows, effectively minimizing the external memory penalty that typically bottlenecks large-scale inference.

Figure 5(b) benchmarks our system against commercial platforms with vendor-optimized software stacks (TensorRT/cuDNN) as a high-performance software-managed baseline. Despite operating with significantly lower peak compute resources and memory bandwidth, our reconfigurable SoC achieves competitive inference latency through efficient heterogeneous integration. Quantified by the system efficiency (latency per token × power)$^{-1}$, our SoC demonstrates superior hardware utilization, yielding 8.12×, 2.56×, and 1.75× higher efficiency than representative edge GPU, server GPU, and server CPU, respectively. Notably, our full SoC operates at 14.23 W total system power which fits well within a typical edge deployment budget, confirming that the efficiency gains stem from coherence-aware orchestration rather than elevated power budgets. These results reinforce our core thesis: System-level reconfigurability—particularly in memory hierarchy and coherence management—offers a viable, scalable solution for Transformer inference on resource-constrained edge platforms without relying on costly hardware over-provisioning.

CONCLUSION

Transformer-based LMs have become fundamental to modern generative AI systems, while their inference demands exceed the capabilities of conventional architecture, especially in resource-constrained edge AI scenarios. In response to this challenge, we present a reconfigurable, coherence-aware SoC architecture that changes system-level parameters, such as accelerator parallelism, memory bandwidth, and coherence mode, during runtime. By leveraging a layer-wise programming model, our architecture tailors the memory subsystem to the distinct arithmetic intensity and data reuse patterns of each Transformer kernel. This approach not only obviates the need for kernel-specific hardware redesigns but also optimizes the Pareto frontier between latency and resource utilization. Ultimately, our reconfigurable SoC achieves 8.12× higher system efficiency compared to a commercial edge GPU, when running a billion-scaled model.

Looking ahead, our future work will focus on two dimensions of scalability. First, we plan to explore inter-kernel pipelining within the accelerator mesh to exploit spatial locality between sequential Transformer layers. Realizing this requires advancements in lightweight synchronization primitives and strict cache residency guarantees to minimize pipeline stalls. Second, we aim to extend our architecture to multichip systems to accommodate working sets that exceed single-device capacity. This introduces

complexities in distributed coherence and nonuniform memory access, necessitating the investigation of remote memory access protocols and coherence-aware NoC designs that ensure seamless scaling without compromising consistency.

In conclusion, our work demonstrates that system-level reconfigurability is a powerful lever for efficient LM inference. By exposing configurable system parameters and embracing runtime adaptability, we offer an alternative path to specialized hardware: one that is more agile, generalizable, and well-suited to the evolving demands of next-generation AI workloads.

ACKNOWLEDGMENTS

This work was supported in part by DOE under Award DE-SC0024458 and in part by the Columbia Center of AI Technology (CAIT).

REFERENCES

1. A. Vaswani et al., "Attention is all you need," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 6000–6010.
2. J. Devlin et al., "BERT: Pre-training of deep bidirectional transformers for language understanding," 2019, *arXiv:1810.04805*.
3. S. Zhang et al., "OPT: Open pre-trained transformer language models," 2022, *arXiv:2205.01068*.
4. S. Kim et al., "Full stack optimization of transformer inference: A survey," 2023, *arXiv:2302.14017*.
5. L. P. Carloni, "From latency-insensitive design to communication-based system-level design," *Proc. IEEE*, vol. 103, no. 11, pp. 2133–2151, Nov. 2015, doi: 10.1109/JPROC.2015.2480849.
6. P. Mantovani et al., "Agile SoC development with open ESP," in *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, 2020, pp. 1–9, doi: 10.1145/3400302.3415753.
7. D. Giri et al., "Accelerators and coherence: An SoC perspective," *IEEE Micro*, vol. 38, no. 6, pp. 36–45, Nov./Dec. 2018, doi: 10.1109/MM.2018.2877288.
8. J. Yang et al., "ReconFormer: A multi-level run-time reconfigurable system-on-chip for accelerating transformers," in *Proc. 35th Int. Conf. Field-Programmable Logic Appl. (FPL)*, 2025, pp. 10–17.
9. H. Shen et al., "Efficient LLM inference on CPUs," in *Proc. Conf. Neural Inf. Process. Systems (NeurIPS)*, 2023, pp. 1–6.
10. "AI Inference." NVIDIA. [Online]. Available: https://developer.nvidia.com/deep-learning-performance-training-inference/ai-inference

JE YANG is a Ph.D. student in computer science at Columbia University, New York, NY, 10027, USA. Her research interests include heterogeneous reconfigurable hardware architecture design for generative artificial intelligence. Yang received her M.S. degree in electrical engineering from the Korea Advanced Institute of Science Technology. Contact her at je.yang@cs.columbia.edu.

GRACEN WALLACE is a software engineer at IBM Research, Yorktown Heights, NY, 10598, USA. Her research interests include machine learning, edge computing and AI hardware. Wallace received her M.S. degree in electrical and computer engineering from the Georgia Institute of Technology. Contact her at gracen@ieee.org.

JOSEPH ZUCKERMAN is a hardware engineer with Jump Trading, Chicago, IL, 60654, USA. His research interests broadly span heterogeneous computing, hardware acceleration, and system-on-chip design and programming. Zuckerman received his Ph.D. degree in computer science from Columbia University, New York, NY, 10027, USA. Contact him at jzuck@cs.columbia.edu.

GABRIELE TOMBESI is a Ph.D. student in computer science at Columbia University, New York, NY, 10027, USA. His research interests include system-on-chip design, design-for-testability, and hardware-software co-design for AI inference. Tombesi received the B.S. degree in physical engineering from the Politecnico di Torino, Turin, Italy, and the joint M.S. degree in electrical engineering from the Politecnico di Torino, and also with the Ecole Polytechnique Federale de Lausanne and Grenoble INP, Lausanne, Switzerland. Contact him at gtombesi@cs.columbia.edu.

LUCA P. CARLONI is professor and chair of computer science at Columbia University, New York, NY, 10027, USA. His research interests include SoC platforms, embedded systems, and open source hardware. Carloni received the Laurea degree (summa cum laude) in electrical engineering from the Universita' di Bologna, Bologna, Italy, in 1995, and the M.S. and Ph.D. degrees in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, CA, USA, in 1997 and 2004, respectively. He is a Fellow of IEEE and ACM. Contact him at luca@cs.columbia.edu.