Combating the Memory Walls: Optimization Pathways for Long-Context Agentic LLM Inference

Haoran Wu*, Can Xiao†, Jiayi Nie*, Xuan Guo†, Binglei Lou†, Jeffrey T.H. Wong†,
Zhiwen Mo†, Cheng Zhang†, Przemyslaw Forys†, Chengyang Ai‡, Timi Adeniran*, Wayne Luk†,
Hongxiang Fan†, Jianyi Cheng‡, Timothy M. Jones*, Rika Antonova*, Robert Mullins*, Aaron Zhao†
*University of Cambridge, †Imperial College London, ‡University of Edinburgh

***Abstract*—Large Language Models (LLMs) serve as the core components of AI agents used across a wide range of applications, including enterprise workflow automation, software engineering, web automation, computer use, and research. These agentic LLM inference tasks are fundamentally different from traditional chatbot-focused inference — they often have much larger context lengths to capture complex, prolonged inputs, such as an entire webpage DOM or complicated tool call trajectories. This, in turn, generates significant off-chip memory traffic for hardware at the inference stage and causes the workload to be constrained by the two memory walls, namely the *bandwidth* and *capacity* walls, preventing the compute units from achieving high utilization.**

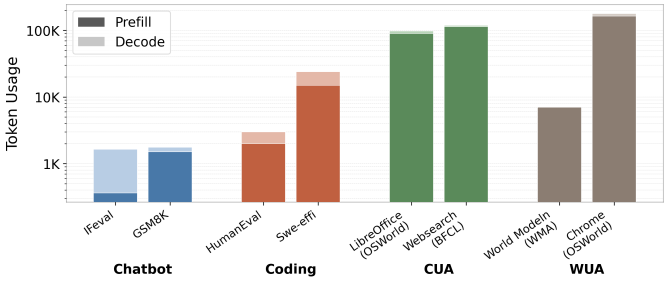
In this paper, we introduce PLENA, a hardware–software co-designed system that applies three core optimization pathways. PLENA features a novel flattened systolic-array architecture (*Pathway 1*) and efficient compute and memory units that support an asymmetric quantization scheme (*Pathway 2*). It also provides native support for FlashAttention (*Pathway 3*). In addition, PLENA is developed with a complete software–hardware stack, including a custom ISA, a compiler, a transaction-level simulator, and an automated design-space exploration flow. Experimental results show that PLENA delivers up to 2.23× and 4.70× higher throughput than the A100 GPU and TPU v6e, respectively, under identical multiplier counts and memory configurations during LLaMA agentic inference. PLENA also achieves up to 4.04× higher energy efficiency than the A100 GPU.

***Index Terms*—LLM Accelerator, Agentic Inference, Systolic Array, FlashAttention, Quantization**

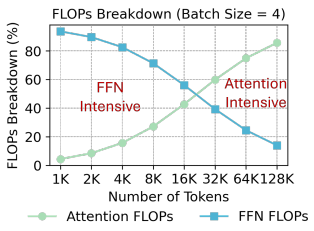
I. Introduction

Transformers have revolutionized AI across various fields, including language, vision, and science [35], [63], [67]. Transformer-based autoregressive large language models (LLMs), like GPT [47] and LLaMA [61], are now widely deployed in many applications, such as chatbots [73], code generation [33] and computer-use workflows [46].

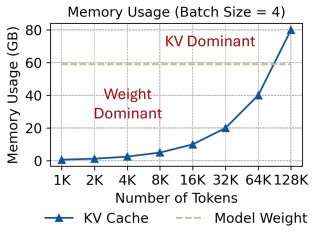
The rapid rise of LLM agentic capabilities, e.g. computer-use [40], web automation [28], [45], and command-line agents [2], relies heavily on their ability to process and reason over very large context windows. For instance, command-line agents must both comprehend and generate large-scale code-bases [31], [51], [71], while tool- and computer-use agentic workflows must track multiple pieces of information across prolonged inputs (e.g. a complete web page DOM), typically requiring very long contexts [11], [16], [36]. Figure 1(a) shows that, when compared to chatbot workloads, agentic workloads consume up to 100× more tokens per inference. In response, modern LLMs have deliberately expanded their context windows: the original GPT-3 [8] supports roughly 2k tokens, whereas GPT-4 [47] reaches up to 32k tokens, and LLAMA-4-Maverick [4] up to 1M tokens.



(a) Token usage comparison across standard chatbot [73], [76], coding [10], [17], and agentic workloads, including Computer Use Agent (CUA) [3], [69] and Web Use Agent (WUA) [9], [69]. Although decode uses far fewer tokens than prefill in agentic tasks, it still contributes significantly to latency due to sequential processing over the full context.



(b) Compute intensity shifts from FFN to attention blocks with an increasing context length.



(c) KV cache scales with context length, eventually dominating memory usage.

Fig. 1: An illustration of agentic inference workloads shows how they typically generate many more tokens per single inference run (a), contain both FFN-compute-intensive and attention-compute-intensive phases (b), and include weight -memory-capacity-dominant and KV-dominant phases (c) within a single inference run.

To clarify the computational characteristics of agentic workloads, Figure 1(b) analyzes a long-context LLAMA-3-70B model. At small context lengths, inference is dominated by Feed-Forward Networks (FFNs), which account for the majority of FLOPs. As the context grows, primarily driven by large prefill sequences in agentic tasks, the computational profile transitions from FFN-intensive to attention-intensive, with attention eventually dominating the overall FLOP count.

Agentic LLM inference also consumes significant HBM resources. Figure 1(c) identifies two major limiting factors on the memory side. First, the large number of KV values and weights that must be read, together with the portion of KV values written back, impose substantial memory-bandwidth demands. Second, as context length increases, the KV-cache requirement grows linearly, quickly increasing memory usage and often surpassing the size of the model weights, making HBM capacity a primary limiting factor. For example, in LLAMA-3-70B, at a 128k context [44], the FP16 KV cache for a single batch is approximately 39 GB, which limits how many batches can be kept on the chip [23]. Building on this observation, we suggest that there are two main challenges on the off-chip memory side, namely, (i) the limited memory bandwidth and (ii) the restricted memory capacity. We collectively term these *memory walls*. Together, they prevent devices from reaching peak performance at inference time, consistent with observations in prior work [15], [23], [74].

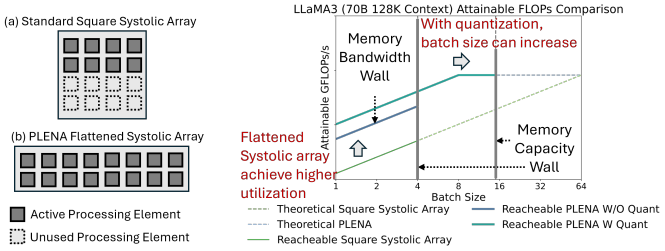
The memory wall phenomenon leads to the underutilization of computing resources on hardware, such as TPUs and GPUs. This effect is particularly evident in General Matrix-Matrix Multiplication (GEMM) operations ($\mathbb{R}^{M\times K} \times \mathbb{R}^{K\times N} \rightarrow \mathbb{R}^{M\times N}$), denoted as $(M, K) \times (K, N)$, which constitute the core computational workload during LLM inference [26]. At the microarchitectural level, most hardware is built with square-shaped systolic arrays or matrix multiplication units, typically designed so that the M and N dimensions are close in size to K. For example, TPU v3 [24] features a 128×128 systolic array, supporting $M = K = N = 128$ GEMM operations. However, in long-context agentic models, as shown in Figure 1(c), memory often becomes the primary constraint for the inference batch size. This results in a *fat GEMM* operation, where the batch-related dimension (typically M in $(M, K) \times (K, N)$) is much smaller than the other operating dimension. This essentially produces an uneven matrix shape[1]. This imbalance hinders systolic arrays and Tensor Cores from achieving a high computational resources utilization rate [29].

To this end, we propose the **P**rogrammable **L**ong-context **E**fficient **N**eural **A**ccelerator (PLENA), an efficient transformer model accelerator system designed to maintain high utilization of GEMM units across all inference stages (prefilling and decoding), particularly for agentic LLM inference tasks with large contexts. PLENA achieves high efficiency for long-context inference by exploring three optimization pathways across both hardware and software design spaces.

First, our novel flattened systolic array (*Pathway 1*) resolves the architectural mismatch caused by the typical square-shaped GEMM used for inference, achieving a higher compute utilization as illustrated in Figures 2(a) and 2(b). Second, we apply an *asymmetric quantization strategy* with Post-Training

[1]All KVs must be stored, so the batch size (the M dimension) is kept lower than the hidden size (K). While various offloading techniques are available [5], they complicate system-level trade-offs and tend to make the system more memory I/O–bound.

[2]64×64 square-shaped systolic array and 8×512 flattened systolic array. Data derived from 144 GB HBM capacity and 512 GB/s memory bandwidth.



(a) PLENA achieves higher utilization than the standard square systolic array with the same resources.

(b) PLENA's optimization pathways—(1) a flattened systolic array and (2) asymmetric quantization—together achieve improved effective memory bandwidth utilization and help reduce memory capacity limitations.

Fig. 2: A comparison of attainable FLOPs between a square-shaped systolic array (e.g. in a TPU) and PLENA's when using the same number of multipliers[2].

Quantization (PTQ) optimizations (*Pathway 2*), where Weights (W) / Activations (A) / KV Cache (KV) can be set to different precisions to alleviate both memory bandwidth and capacity walls. With more aggressive quantization, we free up more space in HBM for data scaling (e.g. supporting larger batch sizes). Figure 2 shows how these pathways together can *increase the utilization* compared to the conventional square-shaped GEMM hardware without any optimization. Finally, as Figure 1(b) shows that attention dominates the compute at longer context lengths, we design PLENA's custom ISA and novel architecture to effectively support FlashAttention (*Pathway 3*)—an IO-aware, fused attention algorithm that substantially reduces off-chip memory traffic [13]. This reduces the likelihood of attention operations saturating memory bandwidth, thereby diminishing the wall's effect.

Together, these optimization pathways yield significantly higher utilization than conventional square-shaped systolic-array accelerators. The main contributions are as follows:

- We analytically characterize the bandwidth and capacity memory walls in agentic LLM inference and show that existing systolic-array accelerators are normally heavily under-utilized when running agentic workloads.
- We introduce three optimization pathways that jointly address the under-utilization caused by memory walls: (i) a flattened systolic array architecture; (ii) an asymmetric quantization scheme, coupled with an in-depth exploration of micro-scaling arithmetic's compatibility with optimization techniques such as rotation and norm-guided iterative optimization; and (iii) a native support for FlashAttention. Together, these enable a holistic approach that addresses both bandwidth and capacity limitations by integrating hardware-level and algorithmic optimizations.
- We present PLENA, a complete hardware–software system that realizes the above optimizations. PLENA integrates: (i) a custom instruction set (PLENA_ISA) for large Transformer inference; (ii) a PyTorch-to-PLENA_ISA compiler; (iii) an HBM-enabled transactional emulator; (iv) an automated, accuracy-aware design-space exploration (DSE)

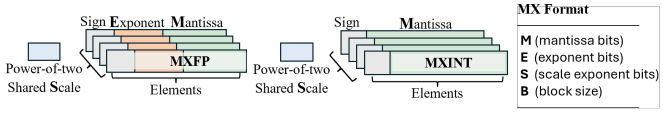


Fig. 3: Illustration of the configurable MX data format design, parameterized with tunable configs. Each block of elements shares a power-of-two scaling factor.

flow; and (v) a full RTL implementation. We demonstrate that PLENA supports different SOTA transformer model variants (e.g. GQA and MLA [42], Dense and MoE [6]). We also show that PLENA achieves superior efficiency for agentic LLM inference. Under identical multiplier counts and memory configurations during LLaMA agentic inference, PLENA delivers up to 2.23× and 4.70× higher throughput than the A100 GPU and TPU v6e, respectively, and up to 4.04× higher energy efficiency (Token / J) than the A100. The entire PLENA system is publicly available[3].

II. Background and Related Work

A. Microscaling Data Formats

The concept of block data representation was introduced to collectively represent groups of values using shared scaling factors [14]. Building on this idea, Rouhani et al. [54] proposed the Microscaling (MX) data format as a specific variant of block data formats, where each block of elements shares a common scale encoded in an `E8M0` power-of-two format. MX formats have since been standardized by the Open Compute Project [52]. Recent extensions explore multi-level scaling, where scaling factors are applied hierarchically across granularities. MicroScopiQ [50] adopts a two-level scaling scheme with coarse block-level and finer micro-block-level scales, while NVFP4 [1] employs a similar hierarchy, using a tensor-level `E8M23` scale and block-level `E4M3` scale. To balance hardware complexity and software performance, we adopt a single-level scaling scheme in our configurable MX data format, with tunable parameters (M, E, S, B) for MXFP and (M, S, B) for MXINT, illustrated in Figure 3.

B. Co-designing PTQ with Microscaling Data Formats

Existing off-the-shelf Post-Training Quantization (PTQ) methods are well-studied for integer data formats [7], [19]. However, we find that these methods are less explored—and in some cases, not directly applicable—to the MX data format.

GPTQ [19] was originally developed for integer quantization. We explore its adaptation to our parameterized MX data formats and propose a variant method that better adapts it to the MX format. Details are deferred to Section IV-B. Rotation-based PTQ methods are among the most effective techniques for mitigating activation outliers. QuaRot [7] demonstrated that the application of the Hadamard transformation can effectively suppress such outliers. However, we empirically experimented and found that without careful treatment, the direct application of these methods can lead to significant model performance degradation for MX data formats. Details are deferred to Section IV-C.

[3] https://github.com/AICrossSim/PLENA

C. FlashAttention

FlashAttention optimizes memory I/O in the standard attention layer [13]. In a standard attention layer, computing $QK^\top$ produces a prohibitively large square matrix, often thousands by thousands in size. Because on-chip memory cannot hold this intermediate result, it must be written to off-chip memory and later reloaded for the subsequent softmax and PV steps, which significantly degrades performance. FlashAttention avoids this round trip by tiling and fusing the attention computation (GEMM–Softmax–GEMM) so that all intermediate results fit on-chip.

Most existing systolic-array–based accelerators do not natively support FlashAttention. SystolicAttention [39] was among the first to integrate FlashAttention into a systolic architecture. In contrast, PLENA adopts a more flexible approach, enabling aggressive memory prefetching overlap and leveraging a mixed-precision supported flattened systolic array with head-level decomposition to achieve higher compute utilization and efficiency.

D. Accelerators and Their Quantization Supports

Recent LLM accelerators [22], [25], [27], [30], [32], [37], [49], [50], [72] explore diverse architectural trade-offs across compute organization, kernel specialization, and system integration. However, many of these designs focus on accelerating specific kernels (e.g. GEMM or attention) rather than supporting the full Transformer inference pipeline, often requiring offloading of unsupported operations to external processors. Such partial coverage can introduce additional data movement and limit sustained utilization under long-context inference workloads. PLENA instead targets full Transformer inference directly on the accelerator fabric.

Prior works have also explored hardware and quantization co-design [25], [27], [30], [50], [70]. MicroScopiQ [50] adopts GPTQ for two-level MX quantization. ANT and MANT [27], [30] propose hybrid data formats that adapt the quantization mode to input distributions at run time. OliVe [25] handles outliers by pairing them with adjacent low-magnitude weights. However, these works mostly focus on weight and activation quantization, without jointly addressing KV cache quantization under long-context inference scenarios. PLENA, by contrast, is the first to natively support tunable MX formats with both hardware-friendly QuaRot [7] and GPTQ [19] while targeting long-context workloads natively.

Prior work, such as SCALE-Sim [56], supports the simulation of flattened systolic arrays for DNN inference, while SARA [55] explores reconfigurable array shapes to optimize DNN workloads. However, these approaches do not explicitly consider the characteristics of autoregressive Transformer inference. PLENA instead adopts a workload-driven design that reshapes the systolic organization to address the imbalance between FlashAttention and FFN computation under memory-constrained batching, as discussed in Section III-B.

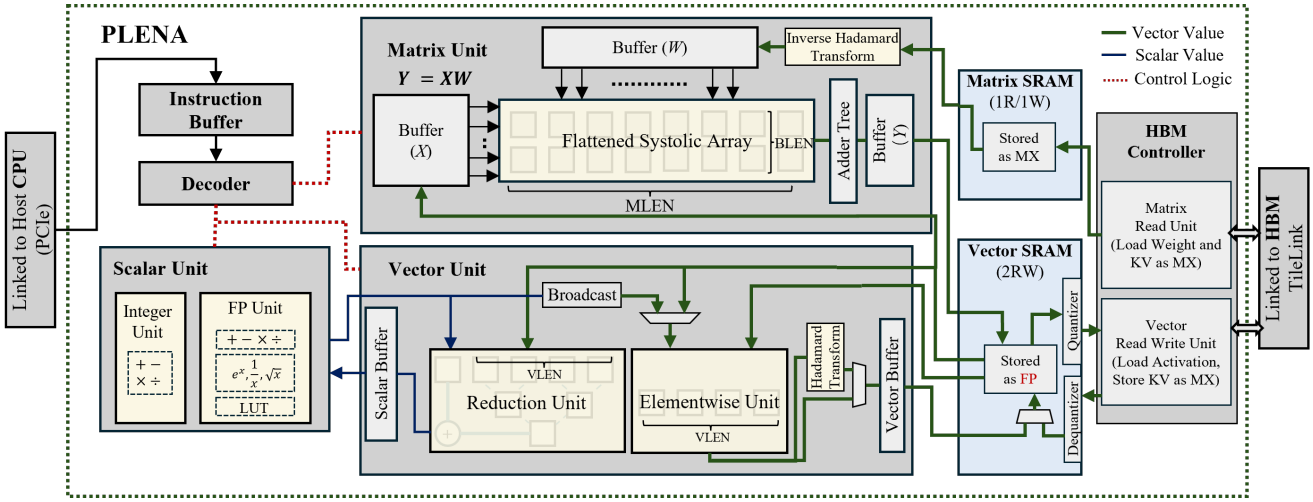


Fig. 4: PLENA accelerator architecture overview. Execution is controlled by the decoder's system-pipeline controller, which derives control signals from decoded instructions and monitors memory dependencies. For example, when reading from a Vector SRAM row that is still being updated by the vector or matrix unit, the controller inserts a stall to ensure correctness.

III. PLENA Hardware System

The overall configuration of PLENA is shown in Figure 4. It employs instruction-level pipelining and mainly consists of three compute units: the Matrix Unit, the Vector Unit, and the Scalar Unit. All units are highly configurable, supporting multiple data types and precisions (Table III), enabling the application of different quantization methods to the accelerator.

PLENA also includes two main on-chip SRAM blocks. The Vector SRAM acts as a scratchpad for computation, storing frequently used data such as activations, which do not need to be written back to HBM, thereby reducing memory access overhead. The custom Matrix SRAM is dedicated to loading weights and KV tensors and supports reading data in either transposed or untransposed access patterns with minimal extra resource cost and access overhead.

A. Asymmetric Arithmetic Data Path

To support asymmetric quantization strategies, PLENA natively supports multiple numeric formats—covering different data types and precisions—across its compute and memory units. This innovative *asymmetric* data-handling configuration has the following characteristics.

(i) Activations are stored in a high-precision floating-point (FP) format on-chip in the Vector SRAM, as they are more sensitive to quantization errors than KV or weights. (ii) KV and weights, being less accuracy-sensitive, can be more aggressively quantized and staged in the Matrix SRAM using lower-precision MX formats (MXFP or MXINT). (iii) An optional on-chip rotation step can suppress outliers before quantization to preserve accuracy.

Furthermore, when appending new K and V vectors to the KV cache in HBM during attention, we selectively apply a Hadamard-based rotation (algorithm detailed in Section IV-C) to suppress outliers before quantizing them to the MX data type and storing them in HBM. Since K and V are consumed exclusively by the attention GEMMs, they are loaded directly into the Matrix SRAM, where the inverse Hadamard transform is applied before use. These rotation/de-rotation stages can be selectively applied per tensor; for example, weights loaded into the matrix unit bypass the inverse transform.

B. Flattened Systolic Array

As shown in Figure 2(b), long-context workloads frequently involve *fat GEMMs* during feed-forward (FFN) computation, where the batch-related dimension (typically M in $(M, K) \times (K, N)$) is much smaller than the others, resulting in uneven matrix shapes (Figure 5), while the reduction dimensions K tend to be very long, for example, the weight–activation GEMM reduces over the model's hidden size (e.g. 4,096 for LLAMA-3-8B and 8,192 for LLAMA-3-70B).

Additionally, in the FlashAttention stage, per-head *fat GEMMs* operations are required. The head dimension is typically small (e.g. 128 for LLAMA-3-70B), and the Grouped Query Attention (GQA) paradigm requires each key head to be multiplied by multiple query heads simultaneously. This results in low utilization of large-scale systolic arrays when performing per-head GEMMs in FlashAttention, as the computation dimension becomes relatively small.

To improve hardware efficiency in the two most computationally intensive layers, we propose a *flattened systolic array* architecture that achieves high utilization for both. For the FFN layer, each processing unit performs a $(\texttt{BLEN}, \texttt{MLEN}) \times (\texttt{MLEN}, \texttt{BLEN})$ GEMM, producing an output of shape $(\texttt{BLEN}, \texttt{BLEN})$. For the FlashAttention module, the systolic array is partitioned into multiple smaller flattened array cores to support per-head GEMM computations, where each core performs a $(\texttt{BLEN}, \texttt{HLEN}) \times (\texttt{HLEN}, \texttt{BLEN})$ GEMM across $(\texttt{MLEN}/\texttt{HLEN})$ heads in parallel.

This flattened systolic array is designed for the output-stationary dataflow in order to maintain a high utilization. As shown in Figure 5, operands stream along the large reduction

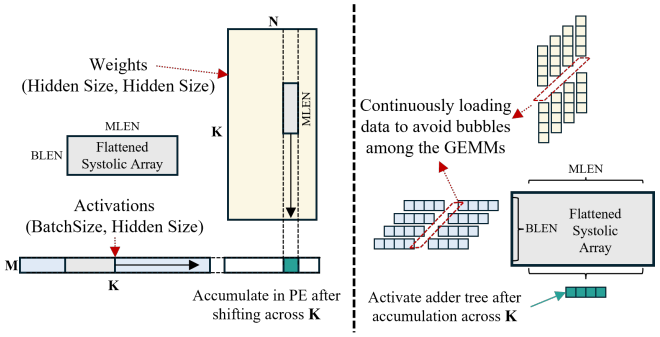


Fig. 5: Processing flow for the weight–activation output stationary GEMM. Because memory capacity constrains batch size, the M dimension remains small. Setting $\texttt{BLEN} = M$ on the flattened systolic array yields high utilization.

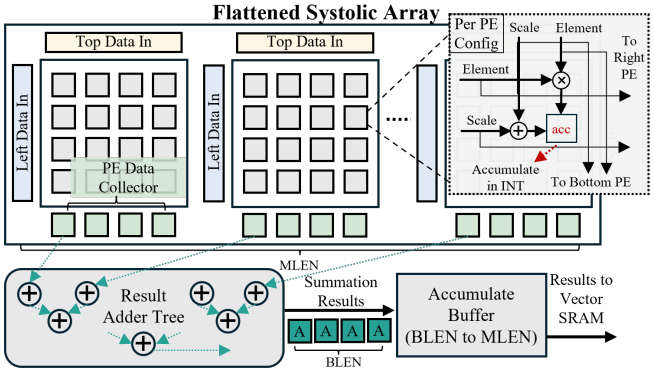


Fig. 6: At each cycle, the flattened systolic array fetches two MLEN-wide inputs: one from the Matrix SRAM (top) and one from the Vector SRAM (left). The inputs are buffered and reordered, then partitioned into MLEN/BLEN subvectors, each of width BLEN. Each subvector is forwarded to a corresponding sub-array from the top and left directions. The scales and elements are streamed separately to each subarray. For improved resource efficiency, each PE consumes MX-format inputs and performs accumulation in INT precision. The accumulated results are converted to the target activation precision before being written back to the Vector SRAM.

dimension K while partial sums remain stationary in the PEs. The array is then fully pipelined, eliminating idling bubbles between consecutive GEMM tiles. The microarchitecture of the flattened systolic array is shown in Figure 6. It is built from a series of small square-shaped systolic arrays (*sub-arrs*), each consisting of a grid of processing elements (PEs). Each PE repeatedly performs multiply–accumulate operations and passes data to its neighboring PEs below and to the right across the array. As described in Section III-A, the systolic array is designed to natively accept data in the MX format.

However, a matrix unit composed solely of *sub-arrs* is insufficient to complete a $(\texttt{BLEN}, \texttt{MLEN}) \times (\texttt{MLEN}, \texttt{BLEN})$ GEMM. Each array accumulates only partial sums for a fragment of the result; producing a complete $(\texttt{BLEN}, \texttt{BLEN})$ output requires a cross-array reduction that sums the partial

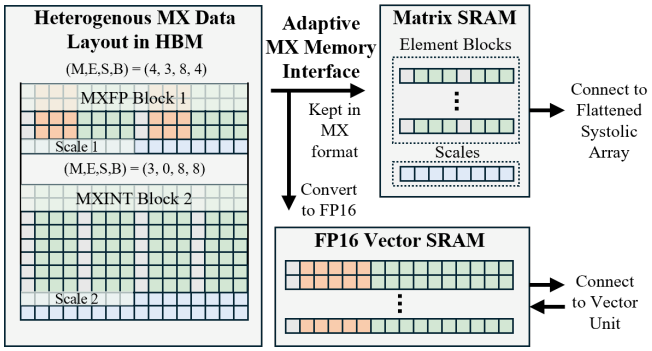


Fig. 7: Data layouts and data paths for the memory system. Data with different MX precisions and datatypes are stored following a unified HBM storage pattern. A conversion to FP16 is performed as the data enter the Vector SRAM, which serves as the scratchpad for the vector unit; the vector unit operates in high-precision FP16. For the Matrix SRAM, MX-formatted data loaded from HBM can be stored directly without additional conversion.

sums held in the PEs across the tiled row. To address this, we integrate a result adder tree (see Figure 6) that performs the cross-array summation efficiently. This unit is invoked via a dedicated instruction M_SUM, as only one cross-array summation is required when computing GEMM along the large reduction dimension. This prevents bubbles and improves computational efficiency.

C. Asymmetric Memory Balancing

Our memory system is characterized by two key properties: 1) Support for asymmetric precisions, variable-length memory transfers, and strided loads/stores to HBM; and 2) Latency hiding for HBM accesses via a memory load unit that operates in parallel with the main execution.

To make more effective use of HBM capacity, as discussed in Section III-A, all data stored in HBM is kept in the MX format. Since concatenating each data block with its per-block scale would rarely yield a combined size that aligns with a power-of-two memory boundary, we instead store the blocks and their corresponding scales for each tensor separately to ensure that both are properly aligned with the memory boundary. This layout improves memory efficiency while maintaining data locality, as illustrated in Figure 7.

The memory load unit is critical for fully utilizing HBM bandwidth. Hardware prefetch engines are integrated into both the Matrix and Vector SRAMs, enabling background fetching from HBM and streaming data into each SRAM while the rest of PLENA continues executing other instructions. This sustains full utilization of the compute unit and avoids stalls due to HBM latency.

D. PLENA ISA

Our customized ISA (Table I) is designed to cover all operations required for transformer inference. Each instruction (32 bits) is structured to balance efficiency and flexibility, enabling support for multiple transformer-based models and

TABLE I: An overview of the PLENA ISA.

Types	Descriptions	# Instr.
Matrix(M)	Controls GEMM and GEMV operations, with or without matrix transposition	12
Vector(V)	Performs elementwise and reduction operations, and rotation for quantization	12
Scalar(S)	Performs scalar INT and FP arithmetic	17
HBM(H)	Handles data transfers between HBM and the Matrix/Vector SRAMs	3
Control(C)	Defines operation settings such as the HBM address, nested-loop configuration, and other execution parameters	8

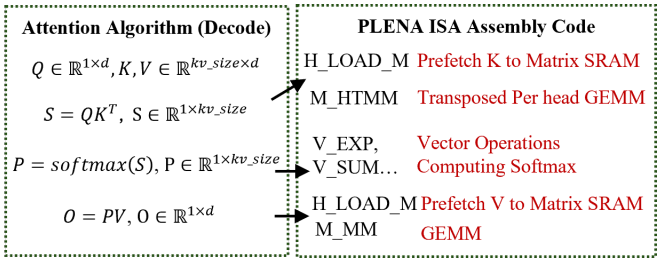


Fig. 8: Example of how the single batch single head attention algorithm maps onto PLENA's custom ISA. Instruction prefixes denote the unit type (e.g., M_ for Matrix instructions).

computation optimizations. In addition to FlashAttention, the ISA supports various transformer variants, including MHA, MLA [42], and MoE [6].

To achieve the efficiency and flexibility balance, the ISA is designed to minimize overhead while maximizing utilization of compute and memory resources. This is achieved through features such as tile-level scheduling, which enables fine-grained control of computation and memory instructions at the tile granularity.

E. Matrix SRAM

The matrix SRAM is designed to support both transposed and non-transposed accesses without additional latency or data movement overhead. This design specifically targets optimizing the transposed matrix multiplication ($QK^\top$) in FlashAttention (see Figure 8) with low hardware overhead.

In autoregressive inference, explicitly transposing large tiles during the ($QK^\top$) computation introduces significant area, energy, and latency overhead. Storing ($K^\top$) directly in HBM is also impractical, as newly generated K vectors must be appended to the KV cache during decoding. Consequently, transposition must be performed on the fly, motivating an SRAM organization that supports both row and column access efficiently without explicit data rearrangement.

As shown in Figure 9, the matrix SRAM distributes each logical row across multiple sub-SRAM banks, storing elements of the same row in different banks at distinct addresses. This layout ensures that row and column accesses map to separate banks, allowing transposed and non-transposed accesses to proceed in parallel without bank conflicts, thereby preserving bandwidth and avoiding explicit data movement.

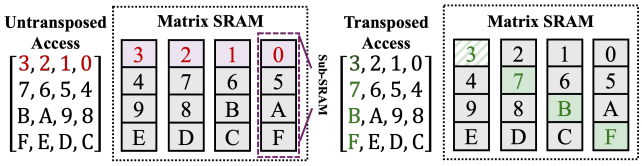


Fig. 9: The transposable matrix SRAM design ensures that, for both untransposed and transposed accesses, each sub-SRAM is accessed by at most one element per cycle. As a result, no additional access cycles are introduced.

We evaluate the proposed design against a conventional transpose-buffer-plus-SRAM baseline with the same read width and memory depth of 32. Using the synthesis tools in Section V-A, our design reduces area by 65.17% while preserving read throughput. It also requires only two cycles for a transpose read—one for sub-SRAM access and one for data reorganization—whereas the baseline must read across 32 rows to produce the transposed output.

F. Supporting FlashAttention

Most existing systolic-array–based accelerators do not natively support FlashAttention due to these four key elements:

1) They do not support tile-level overlapping of off-chip memory prefetching with computation, resulting in additional latency overhead as execution must wait for data to be loaded from off-chip memory.
2) They lack memory-layout support, such as transpose-on-read and efficient strided/blocked streaming.
3) They expose only GEMM primitives and lack in-line, row-wise reductions and nonlinear operations (`max/sum`, `exp`, `div`) required for the online softmax.
4) Their ISAs enforce fixed scheduling and coarse-grained kernel boundaries, which restrict fine-grained tile-by-tile execution and prevent the fused computation pattern.

In PLENA, we address (1) and (2) through the proposed *Matrix SRAM* (see Section III-E), which enables instruction-level control of memory prefetching and supports transpose-on-read with low overhead. Challenge (3) is addressed by vector and scalar units that implement reductions and element-wise operations. The vector width (`VLEN`) is configurable to match the tile dimensions used by FlashAttention. The computation precision is also configurable, but is typically set to higher precision (e.g. FP12) to preserve numerical accuracy during the softmax computation. For (4), our custom ISA provides composable, fine-grained control, enabling persistent, tile-by-tile scheduling of the fused attention pipeline. This allows each stage of FlashAttention to be orchestrated individually at tile granularity. Together, these mechanisms enable PLENA to support FlashAttention natively and efficiently.

G. The PLENA Compilation and Simulation Stack

PLENA provides a comprehensive design and evaluation framework that can rapidly adapt to new models or new hardware accelerators and optimize for them (Figure 10).

Since Transformer computations are highly repetitive and structurally uniform, the PLENA compiler is intentionally

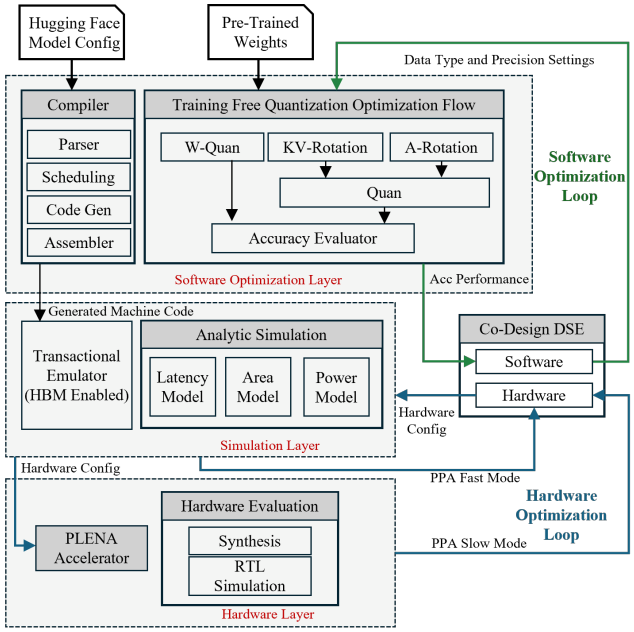


Fig. 10: The co-design framework consists of hierarchical layers (actual hardware, transactional emulator, and analytic simulator) with different fidelities. The transaction-level simulator offers good fidelity (cycle-accurate) while achieving an over 200× speedup compared to RTL simulation.

kept lightweight: it parses configuration metadata from the model configuration file and maps it onto a predefined PLENA custom ISA assembly template.

To evaluate architectural trade-offs, we developed a transaction-level (cycle-approximate) emulator in Rust that executes the generated machine code in an event-driven manner. The emulator models compute execution, instruction scheduling, and memory transactions at cycle granularity. It is integrated with Ramulator [41] and DRAMSys [59] to provide detailed off-chip memory timing and bandwidth modeling, including bank-level behavior. This enables quantitative analysis of memory–compute interaction, which is critical in analysing memory walls in long-context LLM inference.

The emulator supports the full PLENA architectural design space, including asymmetric mixed-precision arithmetic (Section III-A). By bridging analytic modeling and RTL simulation, it enables accurate evaluation of architectural mechanisms—such as flattened systolic mapping and on-chip FlashAttention—while remaining significantly faster than RTL simulation. We validated the simulator against our full RTL implementation: it closely matches the RTL synthesis results in both execution latency and numerical accuracy while delivering roughly a 200× speedup, as shown in Table II.

TABLE II: Average error rates across five trials for different simulation levels, compared with RTL and synthesis results for a single Transformer block of the LLAMA-3-70B model.

Evaluator / Model	Latency	Area	Power	Exe Time
Analytic Simulator	11.32%	4.79%	23.81%	8ms
Transaction Emulator	4.17%	not supp	not supp	4.3mins
RTL Sim. / Synth.	ref	ref	ref	14hrs

IV. QUANTIZATION

Our work is closely related to prior studies that use the microscaling data format [50], [53]. Nonetheless, we highlight in our work that while existing SoTA PTQ optimizations, such as rotation [7] and norm-guided optimization [19], are beneficial for integer quantization, they do not align well with the microscaling format. We identify these caveats for applying PTQ optimization techniques to microscaling arithmetics:

1) For weight quantization, MXFP is generally incompatible with these PTQ optimizations. MXINT demonstrates compatibility, but naïvely applying it leads to degradation. We introduce a novel block-wise clipping optimization that naturally complements block-based arithmetic like MXINT (Section IV-B).
2) For activation quantization, rotation schemes such as QuaRot, when naïvely applied, lead to performance degradation for both MXINT and MXFP. A performance boost is realized only when they are selectively applied to activations (Section IV-C).

In summary, we point out that MXINT with PTQ optimization is the de facto approach for weight quantization. Meanwhile, activation quantization can utilize MXINT or MXFP, but rotation should be applied only selectively. The rest of the section elaborates on these optimization strategies and the root causes of incompatibilities, with Section IV-D detailing the integration of these quantizations into the PLENA system to facilitate a software-hardware co-design.

A. Preliminaries

We start by formalizing MX quantization under a single-level scaling scheme using three elements: the MX data format (τ), the scale factor (s), and the zero point (z). The MX data format is defined by a tuple $\tau = (d, b, B)$, where d denotes the datatype, b is its bit-width, and B is the microscaling block size. For example, $\tau = (\texttt{INT}, 4, 16)$ corresponds to an `MXINT4` format with block size $B = 16$, while $\tau = (\texttt{minifloat}, 4, 16)$ corresponds to an `MXFP4` format with the same block size. In both cases, all values within a block share a single block-wise scale factor s and zero point z.

For any data format τ, the set of representable values is bounded to a finite interval, which we denote as

$$\Omega(\tau) = \{\, x \in \mathbb{R} \mid \min_{(d,b)} \le x \le \max_{(d,b)} \,\}. \tag{1}$$

The representable range $[\min_\tau, \max_\tau]$ of integer MX formats (i.e. $d = \texttt{INT}$) is given by

$$\min_\tau = -(2^{b-1} - 1), \qquad \max_\tau = 2^{b-1} - 1. \tag{2}$$

We partition a high-precision tensor $\mathbf{W}$ into blocks $w \in \mathbb{R}^B$ of size B. For each block w, the scaling factor is

$$s = \frac{\max |w|}{\max_\tau}. \tag{3}$$

The zero-point z shifts the range for alignment; we adopt symmetric quantization ($z = 0$) throughout and omit it from subsequent expressions. Quantization then maps w into the target format w_τ as

$$w_\tau = \mathrm{clip}\Big(\mathrm{RTN}\Big(\frac{w}{s}\Big),\ \min_\tau,\ \max_\tau\Big), \tag{4}$$

where $\mathrm{RTN}(\cdot)$ denotes round-to-nearest projection. The corresponding dequantization operator reconstructs an approximation of the original block

$$Q(w;\, s, \tau) = s \cdot w_\tau. \tag{5}$$

B. Optimizing Microscaling Clipping for Weight Quantization

Existing microscaling arithmetic implementations utilize a static clipping strategy, typically using a fixed value as a clipping threshold for each block (see Equation (3)). However, an advantage of employing smaller blocks is the opportunity for more granular control over numerical values. Consequently, we introduce *microscaling block-wise clipping*, a technique that provides a conscious balancing between the clipping overflow error and the underflow errors for inliers.

For the same sliced block w expressed in format τ, with representable range $[\min_\tau, \max_\tau]$ and empirical range $[\min_w, \max_w]$, we introduce a *clipping parameter* $p \in \mathcal{P} \subset [0.5,\, 0.99]$. This parameter shrinks the effective range to $[p\min_w,\, p\max_w]$.

By sweeping over a discrete set $\mathcal{P}$, we can obtain optimal clipping $p^\star$ for a given block

$$p^\star = \arg\min_{p\in\mathcal{P}} \|w - \mathrm{Q}(w;\, p, \tau)\|_2^2. \tag{6}$$

Here $\|\cdot\|_2^2$ denotes the squared Euclidean norm.

Clipping the empirical range introduces a trade-off between the clipping error and the underflow error. This issue is particularly critical for microscaling-based arithmetic, as the block size is relatively small compared to tensor dimensions. Making an optimal selection of clipping ranges can significantly influence performance; in our experiments, optimized clipping improved perplexity by 5.5% on LLAMA-3-8B in 4-bits weights only quantization setting.

Here, we integrate our clipping optimization directly into GPTQ's iterative error propagation flow, and introduce a new *output-norm guided* blockwise clipping search that *minimizes the quantization error of the output block rather than the weight block*. Formally, let $\mathbf{X} \in \mathbb{R}^{M\times K}$ be the inputs, and $\mathbf{W} \in \mathbb{R}^{N\times K}$ be the weights. Given a linear layer $\mathbf{Y} = \mathbf{X}\mathbf{W}^\top$, we slice the weights across the K dimension with block size B (e.g. MLEN in an MX data format τ), yielding block slices $\mathbf{W}_b \in \mathbb{R}^{N\times B}$ to be quantized, and similarly we can have activations across the K dimension $\mathbf{X}_b \in \mathbb{R}^{M\times B}$. Let $\mathcal{P}$ denote the set of admissible clipping percentiles, and let $\mathrm{Q}(\cdot;\, P, \tau)$ denote per-row quantization in data format τ, where $P = (p_1, \ldots, p_N) \in \mathcal{P}^N$ is a collection of row-wise clipping percentiles, our optimization uses an outer loop optimization with the hessian information $\mathbf{H}_F$ to iteratively calibrate the weight value ($W_b{+}= \boldsymbol{\delta}_F$, adapted from GPTQ)

$$\boldsymbol{\delta}_F = -\Big(\mathbf{W}_b - \mathrm{Q}\big(\mathbf{W}_b;\, P_b^\star, \tau\big)\Big)\ \Big([\mathbf{H}_F^{-1}]_{bb}\Big)^{-1} (\mathbf{H}_F^{-1})_{:,b}, \tag{7}$$

where $\mathbf{H}_F = 2\mathbf{X}_F\mathbf{X}_F^\top$. This is combined with a novel inner loop optimization, which is output-norm guided

$$P_b^\star = \arg\min_{P_b\in\mathcal{P}^N} \left\| \mathbf{X}_b\Big(\mathbf{W}_b - \mathrm{Q}(\mathbf{W}_b;\, P_b, \tau)\Big)^\top \right\|_2^2, \tag{8}$$

C. Selectively Rotated Microscaling Data Formats for Activation and KV Quantization

Rotation-based optimization, such as QuaRot [7], tries to smooth the numerical outlier by introducing a rotation matrix, where $\mathbf{X}, \mathbf{W}, \mathbf{H}$ represent the activation, weight, and Hadamard matrix, respectively

$$l_{rot}(\mathbf{X}) = \mathrm{Q}(\mathbf{X}\mathbf{H}) \cdot \mathrm{Q}(\mathbf{H}^{-1}\mathbf{W}). \tag{9}$$

Surprisingly, we notice that applying the rotation to finer-grained weight quantization (e.g. MXINT with small block sizes) actually increases perplexity. Intuitively, weights have smaller dynamic ranges compared to activations. The rotation may be unnecessary since most weight outliers are already captured by the shared exponents.

We then propose a *selective rotation* strategy for activation quantization

$$\begin{aligned} S &= \arg\min_{s\in\mathcal{M}} \sum_{s\in\mathcal{M}} \Delta_{ppl}(l_{rot}^*), \\ l_{rot}^*(\mathbf{X}) &= \mathrm{Q}(\mathbf{X}\mathbf{H}) \cdot \mathbf{H}^{-1} \cdot \mathrm{Q}(\mathbf{W}). \end{aligned} \tag{10}$$

Now S is a set composed of layers from $\mathcal{M}$, and $\Delta_{ppl}(l_{rot}^*)$ reflects the performance improvement due to rotation for each layer l. The objective is to minimize the sum of the performance loss across all layers in $\mathcal{M}$ to select the subset to be included in S. Another critical difference is that when such rotation is applied to activations, we have to apply a multiplication with $\mathbf{H}^{-1}$ at run-time, and PLENA provides a native hardware support for this operation.

D. Asymmetric Quantization and Hardware Co-Design

As discussed earlier, MXINT is the de facto quantization for weights, whereas we now expose a search space for using either MXINT or MXFP for Section IV-C. Also, we have to consider various precision setups and hardware design parameters (e.g. tile sizes, load/write sizes). We established a co-design framework to conduct such explorations supported by PLENA's multi-fidelity simulators, as shown in Figure 10. Our co-design can run at different fidilities as illustrated in Figure 10, but we choose to run at the transactional-level, unless specified otherwise, for both reasonable speed and good fidelity. Table III shows the search space and its related constraints. Our search space considers a range of arithmetic types for A/KV, including MXINT and MXFP, as well as different precision configurations. The result can provide an

asymmetrically quantized PLENA accelerator design upon completion of the search.

To automate finding the optimal hardware design and quantization parameters, we propose to employ active learning for design space exploration (DSE). We also provide the capability for investigating the trade-offs between optimizing different objectives. For this, we employ multi-objective Bayesian optimization (BO) in BOTorch, which allows exploring the Pareto frontier in an active manner. In our case, the objective function has three components: accuracy, latency, and chip area: $\mathbf{f} = \left[f_{\text{accuracy}}(\cdot), f_{\text{latency}}(\cdot), f_{\text{area}}(\cdot)\right]$. The exploration method also accounts for constraints by applying rejection sampling to discard invalid or infeasible candidates. This avoids unnecessary, costly objective evaluations and accelerates convergence of the search. We first conduct experiments on LLAMA3.2-1B to enable rapid iteration, and then extend our evaluation to LLAMA-3-8B. The results are described in Section IV-D.

TABLE III: Selected hardware and quantization parameter co-design search space. Example constraints include: (1) memory bandwidth constraint $\texttt{MLEN} \cdot \texttt{KV\_WIDTH} \leq \textit{MemBandwidth}$; (2) $\texttt{MLEN} \bmod \texttt{BLEN} = 0$; (3) $\texttt{MLEN} \geq \texttt{HLEN} \geq \texttt{BLEN}$.

Parameter	Description	Search Range
BLEN	Tile size of block unit	[2, 4, ..., 64]
MLEN	Tile size of Matrix Unit	[2, 4, ..., 1024]
VLEN	Tile size of Vector Unit	[2, 4, ..., 1024]
M_LOAD	Matrix SRAM load amount from HBM (num of matrices loaded per instruct)	[2, 4, ..., 256]
V_LOAD	Vector SRAM load amount from HBM (num of vectors loaded per iteration)	[2, 4, ..., 256]
V_WRITE	Vector SRAM write amount to HBM (num of vectors written per iteration)	[2, 4, ..., 256]
ACT_WIDTH	Activation precision	MXINT†, MXFP†
KV_WIDTH	Key/Value precision	MXINT†, MXFP†
FP_SETTING	Floating-point precision	FP†

V. EVALUATION

A. Experiment Setup

a) Models and Datasets: We evaluate our quantization framework on popular open-source LLMs, namely LLaMA-2 [62] and LLaMA-3 [44], as well as MoE [6] (e.g. GPT-OSS) and Qwen3 models [60]. Quantization performance is measured in terms of perplexity on WikiText-2 [43], zero-shot accuracy on six downstream tasks through the lm-evaluation-harness [20], and long-context and agentic workloads: code generation (HumanEval [10]), math reasoning (GSM8K-Platinum [64]), and tool-use (BFCL-Web Search Base [48]). All quantization experiments are run on an NVIDIA B200 GPU 180GB, with PyTorch 2.11.0, CUDA 12.8, Transformers 5.5.0, and lm-evaluation-harness 0.4.11.

The hardware experiments are conducted using token usage traces from standard and agentic benchmarks, evaluated with LLAMA-3.3-70B, as shown in Table IV.

b) Quantization Baselines: We compare against several SoTA quantization methods, including software-based approaches targeting GPUs such as GPTQ [19], OmniQuant [58], and QuaRoT [7], as well as approaches used on hardware accelerators such as Atom [75] and MicroscopiQ [50].

TABLE IV: Token usage (prefill/output) across benchmarks: GSM8K [73], BFCL-Web Search Base (BFCL-W) [48], OSWorld LibreOffice (OSWorld-L) [69].

	GSM8K	BFCL-W	OSWorld-L
Prefill (Tokens)	1.4k	114k	90k
Output (Tokens)	0.2k	5k	8k

c) Accelerator Implementation: PLENA is implemented in SystemVerilog RTL. We perform synthesis using the Synopsys Design Compiler with the 7 nm OpenROAD predictive PDK [12]. This helps us to generate area and power estimates at a 1 GHz clock frequency.

d) Accelerator Baselines: Since our baselines, MicroscopiQ [50], FIGNA [32], SystolicAttention [39] and Olive [25], are either not fully open-sourced or cannot be evaluated under a consistent technology node and toolchain, we re-implemented their core components and integrated them into the PLENA system for a fair inference performance comparison. Additionally, DeepScale [57] is used for overall system performance estimation, scaling all designs to the 7 nm process. Detailed area and power of the core units are evaluated using our own implementations.

e) Inference Process: Instead of comparing only with prior accelerator designs, we also evaluate PLENA against high-performance commercial compute platforms, including GPUs (A100 80GB and H100 80GB) and TPUs (v6e-8), to provide a fair and practical comparison. The GPU experiments are conducted in an environment with Ubuntu 22.04, CUDA 12.8, Python 3.11, PyTorch 2.8.0, and vLLM 0.10 V1. The TPU experiments are conducted in an environment with v2-alpha-tpuv6e software.

B. Balancing Area, Latency and Perplexity via Co-design

This section shows the results of our design-space exploration experiments. Figure 11 shows the Empirical Attainment Surfaces (EAS) for the Pareto fronts found when optimizing with LLAMA3.2-1B and LLAMA-3-8B. EAS is a visualization approach well-suited for conveying the uncertainty of the Pareto fronts from multiple runs with different random seeds [18], [34]. Existing tools support visual analysis for two objectives [66], hence we plot EAS for accuracy and latency first. Figure 11 shows that active learning with BoTorch sampler achieves a significantly better tradeoff between latency and perplexity than naive randomized sampling. Tree-Structured Parzen Estimator (TPE) shows more modest gains when optimizing with LLAMA3.2-1B compared to using the BoTorch sampler, thus we focus on the latter for experiments with LLAMA-3-8B.

In Table V, we show our co-design results generated from multi-objective optimization runs. These runs can yield designs featuring various trade-offs along the Pareto frontier, with some naturally incorporating multi-precision and multi-arithmetic elements. The PLENA system facilitates such exploration, thanks to its comprehensive simulation and RTL support for these arithmetic types and precision levels.

TABLE V: Multi-objective search results for configurations from a BoTorch run on LLAMA-3-8B. We showcase four representative design points on the Pareto frontier with different perplexity (↓), latency (seconds ↓), area (μm^2 ↓) trade-offs. The complete empirical attainment surfaces of the multi-objective search are in Figure 11. **Best** results are highlighted.

Parameters									Metrics		
BLEN	MLEN	VLEN	M_LOAD	V_LOAD	V_WRITE	ACT_WIDTH	KV_WIDTH	FP_SETTING	Perplexity ↓	Lat (s) ↓	Area (mm^2) ↓
32	512	128	128	64	256	MXFP_E4M3	MXFP_E3M4	FP_E4M7	6.70	0.137	137.6
32	1024	1024	256	256	128	MXINT_8	MXINT_4	FP_E3M2	6.76	**0.116**	203.4
8	128	32	128	8	256	MXFP_E3M4	MXFP_E3M4	FP_E5M6	**6.54**	0.166	26.45
16	128	16	4	16	64	MXINT_8	MXFP_E4M3	FP_E3M2	6.60	0.174	**23.64**

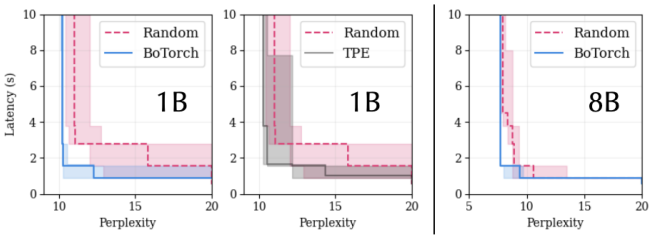


Fig. 11: Empirical Attainment Surfaces for latency (↓) and perplexity (↓) objectives across multiple seeds, evaluated with LLAMA3.2-1B and LLAMA-3-8B over the co-design space shown in Table III. For the 1B model, we run 9 seeds with 50 trials, comparing BoTorch and TPE methods against Random sampling. For the 8B model, we run 5 seeds with 50 trials, comparing BoTorch against Random. Shaded regions show the 25% and 75% attainment bands across seeds.

C. Preserving MX Perplexity via Clipping & Rotation

a) Main Results: We evaluate our quantization method against related work; results are summarized in Table VI. For a fair comparison, we first match prior settings by quantizing only the linear GEMMs in the decoder. Our method matches and outperforms all related work across all three quantization precision settings: W4A16KV16, W4A4KV16, and W4A4KV4. We further report downstream zero-shot results in Table VIII, where our method outperforms QuaRoT [7] across all tasks under the W4A4KV4 setting.We also evaluated the result with the quantized vector cores. We find that quantizing the remaining operators to a MiniFloat E6M5 format is effectively lossless in perplexity while reducing memory footprint by 25% relative to FP16. The key contributions to this performance improvement come from two aspects: 1) **Output-norm guided blockwise clipping search**: by integrating *output-norm guided*, blockwise clipping into iterative weight quantization, we validate that output reconstruction error correlates strongly with end-task performance; consequently, our approach substantially reduces perplexity degradation. 2) **Selective rotation**: Our approach searches for the best layer-wise rotation combination for each model. Unlike QuaRoT [7], which merges rotation into weights, we apply online rotation only to specific layers.

b) Abalation Study: To further assess the impact of our key contributions, we conduct an ablation study on LLAMA-3-8B to validate the effectiveness of 1) output-norm guided blockwise clipping search and 2) selective rotation. The ablation is structured into three stages: (i) *weight-only* quantization, (ii) *activation & KV-cache* quantization on top of quantized weights, and (iii) *full-system* emulation where all MX-aware operators are quantized. We show them in Table VII.

First, MXFP4 always underperforms MXINT4 in all set-

TABLE VI: WikiText-2 perplexity (↓) under GEMM-only emulation (nonlinear ops in full precision) for LLAMA. W/A/KV denote bit widths for weights, activations, and KV cache. Results marked with * are reproduced from released code.

		LLaMA-2 [62]			LLaMA-3 [44]	
Method	**W/A/KV**	**7B**	**13B**	**70B**	**8B**	**70B**
Baseline	16/16/16	5.47	4.83	3.31	6.13	2.85
GPTQ [19]	4/16/16	6.23	5.58	4.28	8.12	3.75
AWQ [38]	4/16/16	5.82	5.19	4.08	7.96	3.58
OmniQuant [58]	4/16/16	5.74	5.02	3.47	7.09	3.46
MicroScopiQ [50]	4/16/16	5.65	5.02	3.42	6.89	**3.25**
QuaRot [7]	4/16/16	5.60	5.00	3.41	6.52*	3.53*
PLENA (MXFP)	4/16/16	7.09	5.91	-	11.95	-
PLENA (ours)	4/16/16	**5.59**	**4.98**	**3.40**	**6.45**	**3.25**
OmniQuant [58]	4/4/16	11.47	8.32	5.41	10.21	5.30
SmoothQuant [68]	4/4/16	20.47	15.63	17.62	29.54	19.32
Atom [75]	4/4/16	6.16	6.12	5.20	8.12	4.69
MicroScopiQ [50]	4/4/16	6.11	5.57	4.48	8.12	4.65
QuaRot [7]	4/4/16	6.02*	5.36*	3.78	8.00*	6.33*
M-ANT [30]	4/4/16	5.92	5.24	-	-	-
PLENA (MXFP)	4/4/16	15.89	10.30	-	91.71	-
PLENA (ours)	4/4/16	**5.82**	**5.14**	**3.56**	**6.99**	**3.87**
QuaRot [7]	4/4/4	6.10	5.40	3.79	8.16	6.66
QuaRot-128G [7]	4/4/4	5.93	5.26	3.61	7.36	5.51
PLENA (MXFP)	4/4/4	67.35	27.44	-	256.22	-
PLENA (ours)	4/4/4	**5.87**	**5.18**	**3.58**	**7.17**	**4.09**

TABLE VII: Ablation study of quantization techniques and their impact on microscaling data formats in LLAMA-3-8B, where the full-system setting quantizes all 9 GEMMs. Results are reported as WikiText-2 perplexity. GPTQ is used for clipping: Err_y denotes output-norm clipping; Err_w denotes weight-norm clipping.

Method	**PPL↓**	**Method**	**PPL↓**
Baseline FP16	6.13	**ACT and KV Only**	
Weight Only		MXFP4	29.75
MXINT + RTN	6.83	MXINT4	7.24
MXFP + RTN	11.94	MXFP4 + Selective Rotate	14.50
MXINT4 + Rotation	6.98	MXINT4 + Selective Rotate	**7.05**
MXFP4 + Rotation	13.71	**MXINT Full System**	
MXINT4 + Err_w Clip	6.53	RTN	9.32
MXINT4 + Err_y Clip	**6.45**	Err_y Clip	8.41
		Err_y Clip +Selective Rotation	**7.43**

TABLE VIII: Zero-shot downstream task accuracy of LLaMA-3 models with 4-bit weight, activation, and KV quantization, evaluated on PIQA (PQ), WinoGrande (WG), HellaSwag (HS), Arc-Easy (A-e), Arc-Challenge (A-c), and LAMBADA (LA).

Method	PQ	WG	HS	A-e	A-c	LA	Avg.
LLaMA-3-8B							
FP16	80.74	72.77	79.06	77.82	53.33	75.63	73.22
QuaRot [7]	75.14	65.82	72.94	68.01	43.34	65.81	65.18
PLENA (ours)	76.99	69.85	75.91	76.73	48.72	72.39	70.10
LLaMA-3-70B							
FP16	84.66	80.51	84.89	85.86	64.25	79.47	79.94
QuaRot	78.07	69.30	77.33	73.44	47.53	69.57	69.21
PLENA (ours)	81.61	77.98	84.12	82.66	58.62	79.24	77.37

TABLE IX: Evaluation of long-context and agentic workloads across code generation (HumanEval [10]), mathematical reasoning (GSM8K-Platinum [64]), and function-calling (BFCL-Web Search Base [48]) benchmarks on Qwen3-32B. W/A/KV denotes bit widths for weights, activations, and KV cache.

Method	W/A/KV	HumanEval pass@1 ↑	GSM8K-PLA EM ↑	BFCL-W Acc ↑
Baseline	16/16/16	89.6	97.85	27.0
Ours	4/4/4	84.1	97.85	24.0

tings. Motivated by this, we adopt MXINT as the default data type for all subsequent evaluations. Second, for weight-only quantization, we show that rotation generally hurts performance—this is simply not compatible with microscaling arithmetic. Furthermore, we demonstrate that our output-norm guided block-wise clipping (Err_y) achieves better performance compared to weight-error guided block-wise clipping (Err_w). Third, selective rotation effectively enhances activation and KV quantization for both MXFP4 and MXINT4. This is different from our observations with weight quantization, where rotation negatively impacts perplexity. We hypothesize this arises from the broader numerical range found in activation and KV values, which benefits from rotation's ability to temper the presence of outliers. Finally, our full system results confirm that both *block-wise clipping search* and *selective activation rotation* improve overall performance.

c) Long-Context and Agentic Workloads: We further validate PLENA's quantization effectiveness on three workloads that stress long-context and agentic settings, showing 4-bit weight, activation, and KV quantization results in Table IX and additional algorithmic ablations in Table X.

D. Improving Utilization via Flattened Systolic Arrays

The utilization analysis of PLENA's flattened systolic array for the FFN and FlashAttention (FA) layers of the LLAMA-3-8B model is summarized in Figure 13. Results for the prefilling stage are omitted because both FFN and FA operate at near-maximum utilization during this phase. For FlashAttention it is excluded as its computation is independent of batch size, while FFN is omitted as its importance diminishes with increasing generated token length.

TABLE X: Ablation study of quantization techniqeus on Qwen3 model across HumanEval, GSM8K-Platinum, and BFCL-Web Search Base. Err_y denotes output-norm clipping. Qwen3-8B on HumanEval and GSM8K-Platinum, Qwen3-32B on BFCL-Web Search Base. Results for HumanEval and GSM8K-Platinum use Qwen3-8B; results for BFCL-Web Search Base use Qwen3-32B, as its higher baseline accuracy better isolates the effect of quantization.

Configuration	HumanEval pass@1 ↑	GSM8K-PLA EM ↑	BFCL-W Acc ↑
Baseline (FP16)	84.8	90.9	27
W-only INT4 (RTN)	82.9	88.7	22
+ ACT & KV INT4	72.0	74.4	15
+ GPTQ	73.2	87.7	24
+ Err_y Clip	74.4	88.6	24
+ Selective Rotation	**78.7**	**88.8**	**24**

TABLE XI: Impact of quantization configurations on memory footprint and bandwidth for LLAMA-3.3-70B under the OSWorld-L workload (90k prefill, 8k output tokens) in Table IV with batch size $B = 8$. W/A/KV denotes the bit precision of weights, activations, and KV cache, respectively.

W/A/KV (bits)	16/16/16	4/16/16	4/4/16	4/4/4
Peak Bandwidth (GB/s)	8192	8192	5120	2048
KV Cache Footprint (GB)	239.26	239.26	239.26	59.81
Weight Storage (GB)	129.46	32.36	32.36	32.36

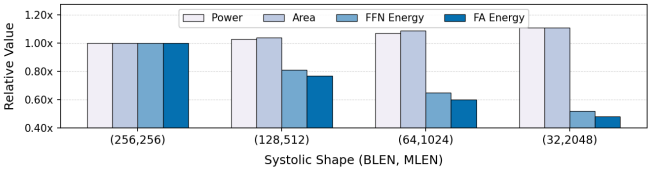


Fig. 12: Power and area comparison of matrix units with different systolic array shapes. Although the flattened systolic array incurs slightly higher area and power, its higher utilization leads to significantly lower effective energy consumption for FFN and attention workloads in the agentic task OSWorld-L.

The DC synthesis results are reported in Table XIII. These results show that the flattened systolic array achieves higher compute-resource utilization for both the FFN and FlashAttention layers compared with prior accelerators. Furthermore, Figure 12 demonstrates that the flattened systolic organization provides higher energy efficiency, despite some power and area overhead compared with conventional square arrays.

The overall ablation study of the systolic-array optimizations is presented in Figure 14. The results show that the flattened systolic array, combined with native FlashAttention support, significantly reduces the execution time of both attention and FFN components across the prefill and decode phases, particularly for long-context inference.

E. System Performance Analysis

The system-level performance comparison is shown in Table XII, evaluating both small and large GQA-based LLaMA

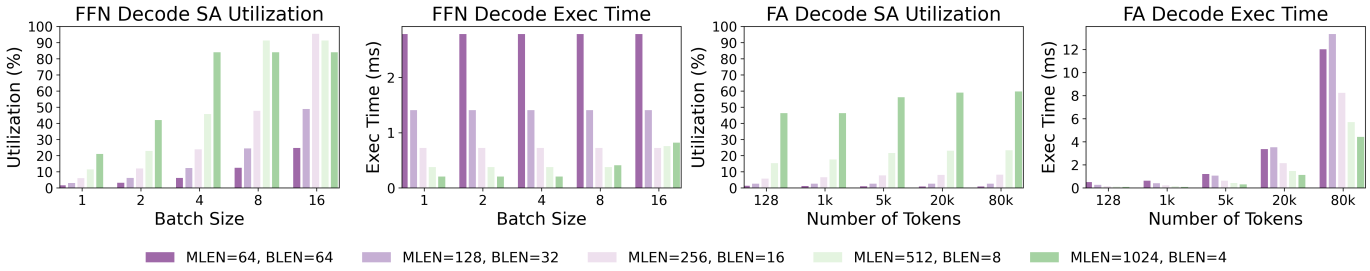


Fig. 13: The systolic array reaches optimal utilization in the FFN layer when its block length (BLEN) aligns with the batch size. FA = Flash Attention. SA = systolic array. For FA, flattening the array enhances utilization by allowing parallel processing of multiple attention heads, and is particularly efficient for long-context inference with smaller effective batch sizes.

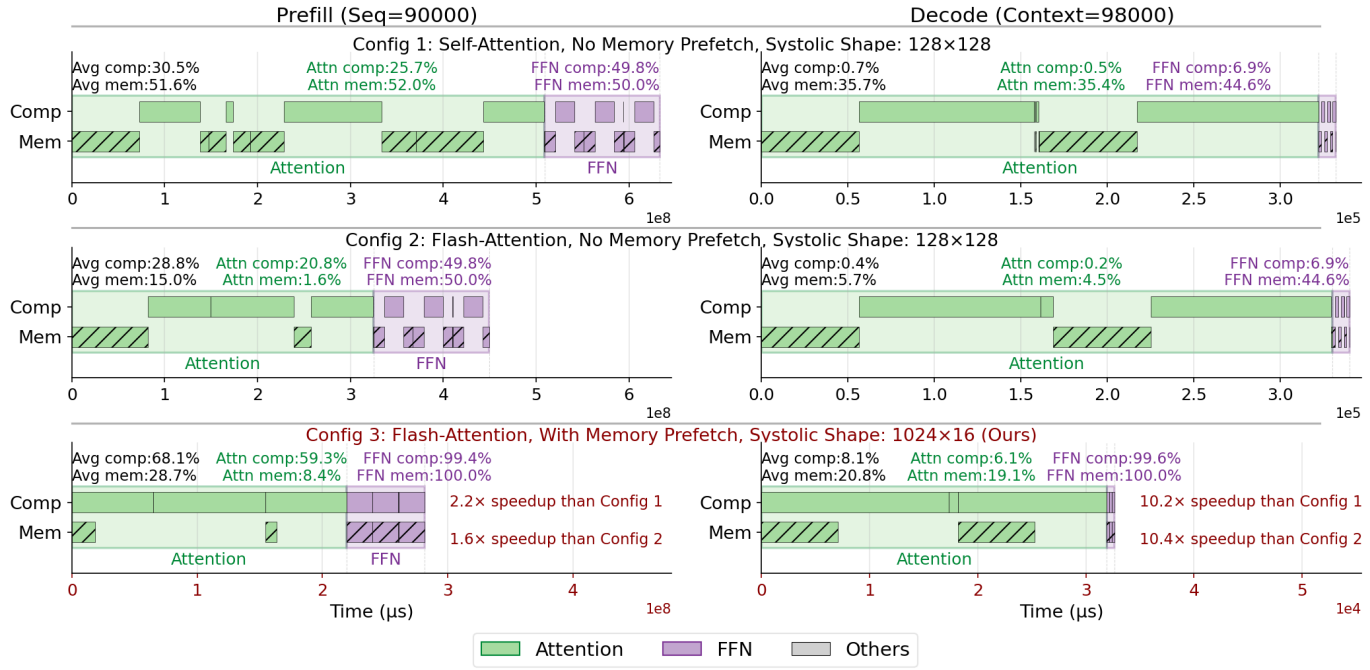


Fig. 14: This figure shows the timing performance breakdown of PLENA across the prefill and decode stages for the LLAMA-3.3-70B model with a batch size of 16. The breakdown includes compute active time (Comp), memory active time (Mem), systolic array (SA) utilization, and memory bandwidth utilization across the overall inference flow. With on-chip FlashAttention support, large intermediate activations are retained on-chip rather than written to off-chip memory, substantially reducing memory traffic, while memory prefetching hides most data-access latency. In addition, the flattened systolic array configuration maintains high utilization across both prefill and decode stages. For attention workloads, the flattened array achieves high compute and memory utilization by enabling parallel multi-head execution and head preloading.

models as well as the MoE-based GPT-OSS model and Qwen3-32B and supporting long-context inputs. The performance results for PLENA and MicroScopiQ are obtained using our analytic simulator. For fairness, we conduct a system-level comparison against a 4×A100 SXM GPU system (80 GB HBM and 1.99 TB/s bandwidth per GPU), a 4×H100 SXM GPU system (80 GB HBM and 3.35 TB/s bandwidth per GPU), and a 16×TPU v6e system (32 GB HBM and 1.56 TB/s bandwidth per device). Both PLENA and MicroScopiQ are modeled as 16-accelerator systems with aggregate HBM capacity and bandwidth equivalent to A100 GPU systems.

To account for GPUs' non-compute components, we determine the number of devices by approximately matching multiplier counts rather than silicon area, given differences in fabrication nodes. Furthermore, for H100, we align memory capacity instead of multiplier counts, as it provides substantially higher compute resources than the other platforms. The co-design-selected PLENA configuration (BLEN = 32, MLEN = 2048, VLEN = 2048, Precision W/A/KV = 4/4/4) demonstrates improved performance across all evaluated workloads.

As shown, PLENA achieves higher TPS than both the A100 and TPU v6e under identical HBM settings and multiplier counts, reaching up to 2.23× that of the A100 and 4.70× that of the TPU v6e for agentic workload. The higher TTFT

TABLE XII: System-level comparison across workloads in Table IV. Performance evaluation occurs under full HBM-capacity utilization, setting the batch size (BS) to the largest fitting value per workload-hardware pair. Note: We reproduced MicroScopiQ [50] and deployed its compute unit on the PLENA platform for testing. And for GPT-OSS 20B (MoE) [6] and Qwen3-32B [60], the remaining accelerators and TPUs are not included since they do not support these configurations [65].

	LLAMA-3.1-8B															
	(1.4k, 0.2k)				**(114k, 5k)**				**(90k, 8k)**				**(90k, 8k) Equal Batch**			
System	TTFT (s)	TPS (×A100)	Tok/J	BS	TTFT (s)	TPS (×A100)	Tok/J	BS	TTFT (s)	TPS (×A100)	Tok/J	BS	TTFT (s)	TPS (×A100)	Tok/J	BS
A100	0.68	1.00x	1.00x	2048	7.40	1.00x	1.00x	16	5.00	1.00x	1.00x	16	5.00	1.00x	1.00x	16
A100 QuaRot [7]	0.73	1.12x	1.12x	4096	8.63	1.10x	1.10x	32	5.97	1.14x	1.14x	32	4.79	1.08x	1.08x	16
H100	2.42	1.65x	0.94x	2048	2.66	2.50x	1.43x	16	1.83	2.48x	1.41x	16	1.83	2.48x	1.41x	16
H100 QuaRot [7]	2.51	1.77x	1.01x	4096	2.97	2.57x	1.47x	32	2.01	2.55x	1.46x	32	1.77	2.51x	1.43x	16
TPU v6e	5.61	0.88x	N/A	2048	7.58	0.51x	N/A	16	7.23	0.53x	N/A	16	7.23	0.53x	N/A	16
MicroScopiQ [50]	3.47	0.83x	1.67x	8192	21.28	0.37x	0.74x	64	19.13	0.39x	0.78x	64	4.93	0.27x	0.54x	16
PLENA	3.41	**1.91x**	**3.50x**	8192	20.13	1.45x	**2.66x**	64	18.87	1.45x	**2.65x**	64	4.68	1.17x	**2.10x**	16
	LLAMA-3.3-70B															
	(1.4k, 0.2k)				**(114k, 5k)**				**(90k, 8k)**				**(90k, 8k) Equal Batch**			
System	TTFT (s)	TPS (×A100)	Tok/J	BS	TTFT (s)	TPS (×A100)	Tok/J	BS	TTFT (s)	TPS (×A100)	Tok/J	BS	TTFT (s)	TPS (×A100)	Tok/J	BS
A100	0.78	1.00x	1.00x	256	43.18	1.00x	1.00x	4	29.67	1.00x	1.00x	4	29.67	1.00x	1.00x	4
A100 QuaRot [7]	1.17	1.08x	1.08x	512	42.89	1.13x	1.13x	8	32.17	1.13x	1.13x	8	27.69	1.11x	1.11x	4
H100	0.34	2.34x	1.34x	256	14.30	2.13x	1.21x	4	10.10	2.04x	1.22x	4	10.10	2.04x	1.22x	4
H100 QuaRot [7]	0.44	2.36x	1.35x	512	16.12	2.19x	1.25x	8	11.37	2.14x	1.22x	8	9.88	2.08x	1.18x	4
TPU v6e	11.7	0.85x	N/A	256	41.96	0.46x	N/A	4	37.61	0.47x	N/A	4	37.61	0.47x	N/A	4
MicroScopiQ [50]	8.32	0.79	1.59x	1024	73.28	0.20x	0.41x	16	49	0.17x	0.35x	16	23.93	0.11x	0.23x	4
PLENA	7.58	1.82x	**3.32x**	1024	69.10	**2.23x**	**4.07x**	16	43.43	**2.21x**	**4.04x**	16	21.68	1.34x	**2.45x**	4
	GPT-OSS 20B (MoE)															
	(1.4k, 0.2k)				**(114k, 5k)**				**(90k, 8k)**				**(90k, 8k) Equal Batch**			
System	TTFT (s)	TPS (×A100)	Tok/J	BS	TTFT (s)	TPS (×A100)	Tok/J	BS	TTFT (s)	TPS (×A100)	Tok/J	BS	TTFT (s)	TPS (×A100)	Tok/J	BS
A100	1.46	1.00x	1.00x	1024	11.81	1.00x	1.00x	8	8.05	1.00x	1.00x	8	8.05	1.00x	1.00x	8
H100	4.03	0.89x	0.51x	1024	1.85	3.10x	1.78x	8	1.38	2.90x	1.66x	8	1.38	2.90x	1.66x	8
PLENA	13.41	**1.15x**	**2.10x**	4096	47.63	1.96x	**3.58x**	64	41.08	1.93x	**3.52x**	64	9.77	0.99x	**1.79x**	8
	Qwen3-32B															
	(1.4k, 0.2k)				**(114k, 5k)**				**(90k, 8k)**				**(90k, 8k) Equal Batch**			
System	TTFT (s)	TPS (×A100)	Tok/J	BS	TTFT (s)	TPS (×A100)	Tok/J	BS	TTFT (s)	TPS (×A100)	Tok/J	BS	TTFT (s)	TPS (×A100)	Tok/J	BS
A100	0.88	1.00x	1.00x	1024	28.90	1.00x	1.00x	8	19.19	1.00x	1.00x	8	19.19	1.00x	1.00x	8
H100	1.19	2.13x	1.22x	1024	9.24	2.29x	1.31x	8	6.29	2.21x	1.26x	8	6.29	2.21x	1.26x	8
PLENA	4.38	1.40x	**2.56x**	4096	108.1	1.22x	**2.23x**	64	90.71	1.23x	**2.25x**	64	23.14	1.14x	**2.08x**	8

TABLE XIII: Compute area, utilization, and attainable FLOPs for different systolic-array designs. Baselines use a 64×64 array, while PLENA uses a flattened 4×1024 array. *S.A.T* denotes standard attainable TOPs on GSM8K, and *A.A.T* denotes agentic attainable TOPs on the OSWorld-L workload from Table IV, measured across the full inference flow. SystolicAttn has a larger compute area because it uses FP16 without quantization, but achieves high utilization by fusing the full attention computation within the array.

Design	Comp Area (mm^2)	TOPs/mm^2	S.A.T/mm^{2*}	A.A.T/mm^{2*}
MicroscopiQ [50]	0.1378	59.45	26.36	5.83
Olive [25]	0.319	25.66	13.76	2.40
FIGNA [32]	0.471	17.39	7.51	1.83
SystolicAttn [39]	1.17	14.00	7.14	4.76
PLENA	0.237	34.49	**29.31**	**12.81**

observed in PLENA is explained by its ability to store more batches within the same HBM capacity using our quantization scheme. As batch size increases, the prefill stage grows longer due to additional memory accesses and computation.

VI. Conclusion

We present PLENA, a novel accelerator design system that exploits a flattened systolic array for efficient agentic inference acceleration. We identify the underutilization challenges posed by memory bandwidth and capacity walls for agentic model inference. In order to address them, we propose an asymmetric quantization scheme for hardware acceleration in MX and native architectural support for FlashAttention. Beyond the hardware, PLENA also presents a full system exploration framework, including new ISA support, automated code generation, multi-level simulators, and a co-design exploration engine. This provides an exploration platform beyond a specific accelerator architecture implementation and enables future research to prototype and explore optimizations for emerging transformer models, similar to Berkeley's Gemmini framework for DNNs [21].

Our future work will focus on integrating PLENA with GPU systems to enable heterogeneous LLM acceleration, leveraging the strengths of both architectures. Furthermore, as multi-turn and multi-modal agentic workloads become increasingly prevalent, we plan to extend PLENA to better support and optimize for these emerging workloads.

Acknowledgment

This work was supported by the Advanced Research and Invention Agency under the Scaling Compute Programme.

REFERENCES

[1] F. Abecassis, A. Agrusa, D. Ahn, J. Alben, S. Alborghetti, M. Andersch, S. Arayandi, A. Bjorlin, A. Blakeman, E. Briones *et al.*, "Pretraining Large Language Models with NVFP4," *arXiv preprint arXiv:2509.25149*, 2025.

[2] M. Agarwal, J. J. Barroso, T. Chakraborti, E. M. Dow, K. Fadnis, B. Godoy, M. Pallan, and K. Talamadupula, "Project CLAI: Instrumenting the Command Line as a New Environment for AI Agents," 2020. [Online]. Available: https://arxiv.org/abs/2002.00762

[3] S. Agashe, K. Wong, V. Tu, J. Yang, A. Li, and X. E. Wang, "Agent S2: A Compositional Generalist-Specialist Framework for Computer Use Agents," 2025. [Online]. Available: https://arxiv.org/abs/2504.00906

[4] M. AI. (2025, Apr.) The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation. Accessed: 2025-08-16. [Online]. Available: https://ai.meta.com/blog/llama-4-multimodal-intelligence/

[5] R. Y. Aminabadi, S. Rajbhandari, M. Zhang, A. A. Awan, C. Li, D. Li, E. Zheng, J. Rasley, S. Smith, O. Ruwase, and Y. He, "DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale," 2022. [Online]. Available: https://arxiv.org/abs/2207.00032

[6] M. Artetxe, S. Bhosale, N. Goyal, T. Mihaylov, M. Ott, S. Shleifer, X. V. Lin, J. Du, S. Iyer, R. Pasunuru, G. Anantharaman, X. Li, S. Chen, H. Akin, M. Baines, L. Martin, X. Zhou, P. S. Koura, B. O'Horo, J. Wang, L. Zettlemoyer, M. Diab, Z. Kozareva, and V. Stoyanov, "Efficient Large Scale Language Modeling with Mixtures of Experts," 2022. [Online]. Available: https://arxiv.org/abs/2112.10684

[7] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, P. Cameron, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, "QuaRot: Outlier-Free 4-Bit Inference in Rotated LLMs," 2024. [Online]. Available: https://arxiv.org/abs/2404.00456

[8] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[9] H. Chae, N. Kim, K. T. iunn Ong, M. Gwak, G. Song, J. Kim, S. Kim, D. Lee, and J. Yeo, "Web Agents with World Models: Learning and Leveraging Environment Dynamics in Web Navigation," 2025. [Online]. Available: https://arxiv.org/abs/2410.13232

[10] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating Large Language Models Trained on Code," 2021.

[11] T. L. S. D. Chezelles, M. Gasse, A. Drouin, M. Caccia, L. Boisvert, M. Thakkar, T. Marty, R. Assouel, S. O. Shayegan, L. K. Jang, X. H. Lù, O. Yoran, D. Kong, F. F. Xu, S. Reddy, Q. Cappart, G. Neubig, R. Salakhutdinov, N. Chapados, and A. Lacoste, "The BrowserGym Ecosystem for Web Agent Research," 2025. [Online]. Available: https://arxiv.org/abs/2412.05467

[12] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, Jul. 2016.

[13] T. Dao, "FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning," 2023. [Online]. Available: https://arxiv.org/abs/2307.08691

[14] B. Darvish Rouhani, R. Zhao, V. Elango, R. Shafipour, M. Hall, M. Mesmakhosroshahi, A. More, L. Melnick, M. Golub, G. Varatkar *et al.*, "With shared microexponents, a little shifting goes a long way," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[15] M. Davies, N. Crago, K. Sankaralingam, and C. Kozyrakis, "Efficient LLM Inference: Bandwidth, Compute, Synchronization, and Capacity are all you need," 2025. [Online]. Available: https://arxiv.org/abs/2507.14397

[16] A. Drouin, M. Gasse, M. Caccia, I. H. Laradji, M. Del Verme, T. Marty, D. Vazquez, N. Chapados, and A. Lacoste, "WorkArena: How Capable are Web Agents at Solving Common Knowledge Work Tasks?" in *Proceedings of the 41st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, Eds., vol. 235. PMLR, 21–27 Jul 2024, pp. 11 642–11 662. [Online]. Available: https://proceedings.mlr.press/v235/drouin24a.html

[17] Z. Fan, K. Vasilevski, D. Lin, B. Chen, Y. Chen, Z. Zhong, J. M. Zhang, P. He, and A. E. Hassan, "SWE-Effi: Re-Evaluating Software AI Agent System Effectiveness Under Resource Constraints," 2025. [Online]. Available: https://arxiv.org/abs/2509.09853

[18] C. M. Fonseca, A. P. Guerreiro, M. López-Ibánez, and L. Paquete, "On the computation of the empirical attainment function," in *International Conference on Evolutionary Multi-criterion Optimization*. Springer, 2011, pp. 106–120.

[19] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[20] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou, "The language model evaluation harness," 07 2024. [Online]. Available: https://zenodo.org/records/12608602

[21] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration," in *Proceedings of the 58th Annual Design Automation Conference (DAC)*, 2021.

[22] S. Ghodrati, S. Kinzer, H. Xu, R. Mahapatra, Y. Kim, B. H. Ahn, D. K. Wang, L. Karthikeyan, A. Yazdanbakhsh, J. Park, N. S. Kim, and H. Esmaeilzadeh, "Tandem Processor: Grappling with Emerging Operators in Neural Networks," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1165–1182. [Online]. Available: https://doi.org/10.1145/3620665.3640365

[23] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "AI and Memory Wall," *IEEE Micro*, vol. 44, no. 3, p. 33–39, May 2024. [Online]. Available: https://doi.org/10.1109/MM.2024.3373763

[24] Google, "System Architecture: TPU VM," https://cloud.google.com/tpu/docs/system-architecture-tpu-vm, Google Cloud, Technical Report, 2025, last updated August 1, 2025.

[25] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[26] C. Guo, C. Wei, J. Tang, B. Duan, S. Han, H. Li, and Y. Chen, "Transitive Array: An Efficient GEMM Accelerator with Result Reuse," 2025. [Online]. Available: https://arxiv.org/abs/2504.16339

[27] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Ant: Exploiting adaptive numerical data type for low-bit deep neural network quantization," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1414–1433.

[28] H. He, W. Yao, K. Ma, W. Yu, Y. Dai, H. Zhang, Z. Lan, and D. Yu, "WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models," 2024. [Online]. Available: https://arxiv.org/abs/2401.13919

[29] K. Hong, G. Dai, J. Xu, Q. Mao, X. Li, J. Liu, K. Chen, Y. Dong, and Y. Wang, "FlashDecoding++: Faster Large Language Model Inference on GPUs," 2024. [Online]. Available: https://arxiv.org/abs/2311.01282

[30] W. Hu, H. Zhang, C. Guo, Y. Feng, R. Guan, Z. Hua, Z. Liu, Y. Guan, M. Guo, and J. Leng, "M-ANT: Efficient Low-bit Group Quantization for LLMs via Mathematically Adaptive Numerical Type," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2025, pp. 1112–1126.

[31] Y. Ishibashi and Y. Nishimura, "Self-Organized Agents: A LLM Multi-Agent Framework toward Ultra Large-Scale Code Generation and Optimization," 2024. [Online]. Available: https://arxiv.org/abs/2404.02183

[32] J. Jang, Y. Kim, J. Lee, and J.-J. Kim, "FIGNA: Integer Unit-Based Accelerator Design for FP-INT GEMM Preserving Numerical Accuracy," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 760–773.

[33] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A Survey on Large Language Models for Code Generation," 2024. [Online]. Available: https://arxiv.org/abs/2406.00515

[34] J. Knowles, "A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers," in *5th International Conference on Intelligent Systems Design and Applications (ISDA'05)*. IEEE, 2005, pp. 552–557.

[35] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large Language Models are Zero-Shot Reasoners," 2023. [Online]. Available: https://arxiv.org/abs/2205.11916

[36] D. Lee, J. Lee, K. Kim, J. Tack, J. Shin, Y. W. Teh, and K. Lee, "Learning to Contextualize Web Pages for Enhanced Decision Making by LLM Agents," 2025. [Online]. Available: https://arxiv.org/abs/2503.10689

[37] J. Lee, W. Lee, and J. Sim, "Tender: Accelerating Large Language Models via Tensor Decomposition and Runtime Requantization," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 1048–1062.

[38] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 87–100, 2024.

[39] J. Lin, G. Chen, Y. Li, and T. Bourgeat, "SystolicAttention: Fusing FlashAttention within a Single Systolic Array," 2025. [Online]. Available: https://arxiv.org/abs/2507.11331

[40] Y. Lu, J. Yang, Y. Shen, and A. Awadallah, "OmniParser for Pure Vision Based GUI Agent," 2024. [Online]. Available: https://arxiv.org/abs/2408.00203

[41] H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkçı, and O. Mutlu, "Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator," 2023. [Online]. Available: https://arxiv.org/abs/2308.11030

[42] F. Meng, P. Tang, X. Tang, Z. Yao, X. Sun, and M. Zhang, "TransMLA: Multi-Head Latent Attention Is All You Need," 2025. [Online]. Available: https://arxiv.org/abs/2502.07864

[43] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[44] A. Meta, "Introducing meta llama 3: The most capable openly available llm to date," *Meta AI*, 2024.

[45] M. Müller and G. Žunič, "Browser Use: Enable AI to control your browser," https://github.com/browser-use/browser-use, 2024, gitHub repository.

[46] B. Niu, Y. Song, K. Lian, Y. Shen, Y. Yao, K. Zhang, and T. Liu, "Flow: Modularized Agentic Workflow Automation," 2025. [Online]. Available: https://arxiv.org/abs/2501.07834

[47] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Łukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Łukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph, "GPT-4 Technical Report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[48] S. G. Patil, H. Mao, F. Yan, C. C.-J. Ji, V. Suresh, I. Stoica, and J. E. Gonzalez, "The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models," in *Forty-second International Conference on Machine Learning*, 2025.

[49] J. Qin, T. Xia, C. Tan, J. Zhang, and S. Q. Zhang, "PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 845–861. [Online]. Available: https://doi.org/10.1145/3676641.3716013

[50] A. Ramachandran, S. Kundu, and T. Krishna, "Microscopiq: Accelerating foundational models through outlier-aware microscaling quantization," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1193–1209.

[51] S. Rando, L. Romani, A. Sampieri, L. Franco, J. Yang, Y. Kyuragi, F. Galasso, and T. Hashimoto, "LongCodeBench: Evaluating Coding LLMs at 1M Context Windows," 2025. [Online]. Available: https://arxiv.org/abs/2505.07897

[52] B. Rouhani, N. Garegrat, T. Savell, R. Zhao, and A. More, "OCP Microscaling Formats (MX) Specification," *Open Compute Project*, 2023.

[53] B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf, S. Dusan, V. Elango, M. Golub, A. Heinecke, P. James-Roxby, D. Jani, G. Kolhe, M. Langhammer, A. Li, L. Melnick, M. Mesmakhosroshahi, A. Rodriguez, M. Schulte, R. Shafipour, L. Shao, M. Siu, P. Dubey, P. Micikevicius, M. Naumov, C. Verrilli, R. Wittig, D. Burger, and E. Chung, "Microscaling Data Formats for Deep Learning," 2023. [Online]. Available: https://arxiv.org/abs/2310.10537

[54] B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf *et al.*, "Microscaling data formats for deep learning," *arXiv preprint arXiv:2310.10537*, 2023.

[55] A. Samajdar, E. Qin, M. Pellauer, and T. Krishna, "Self adaptive reconfigurable arrays (SARA): learning flexible GEMM accelerator configuration and mapping-space using ML," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 583–588. [Online]. Available: https://doi.org/10.1145/3489517.3530506

[56] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "SCALE-Sim: Systolic CNN Accelerator Simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[57] S. Sarangi and B. Baas, "DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[58] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo, "Omniquant: Omnidirectionally calibrated quantization for large language models," *arXiv:2308.13137*, 2023.

[59] L. Steiner, M. Jung, F. S. Prado, K. Bykov, and N. Wehn, "DRAMSys4.0: A Fast and Cycle-Accurate SystemC/TLM-Based DRAM Simulator," in *Embedded Computer Systems: Architectures, Modeling, and Simulation: 20th International Conference, SAMOS 2020, Samos, Greece, July 5–9, 2020, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 110–126. [Online]. Available: https://doi.org/10.1007/978-3-030-60939-9_8

[60] Q. Team, "Qwen3 Technical Report," 2025. [Online]. Available: https://arxiv.org/abs/2505.09388

[61] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and Efficient Foundation Language Models," 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[62] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[63] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[64] J. Vendrow, E. Vendrow, S. Beery, and A. Madry, "Do large language model benchmarks test reliability?" *arXiv preprint arXiv:2502.03461*, 2025.

[65] vLLM Project, "Hardware Supported Models – TPU (Text-Only Language Models)," https://docs.vllm.ai/en/v0.9.2/models/hardware_supported_models/tpu.html#text-only-language-models, 2025, accessed: 2025-11-12.

[66] S. Watanabe, "Python tool for visualizing variability of Pareto fronts over multiple runs," *arXiv preprint arXiv:2305.08852*, 2023.

[67] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, "Emergent Abilities of Large Language Models," 2022. [Online]. Available: https://arxiv.org/abs/2206.07682

[68] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 087–38 099.

[69] T. Xie, D. Zhang, J. Chen, X. Li, S. Zhao, R. Cao, T. J. Hua, Z. Cheng, D. Shin, F. Lei, Y. Liu, Y. Xu, S. Zhou, S. Savarese, C. Xiong, V. Zhong, and T. Yu, "OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments," 2024. [Online]. Available: https://arxiv.org/abs/2404.07972

[70] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 811–824.

[71] D. Zan, Z. Huang, W. Liu, H. Chen, L. Zhang, S. Xin, L. Chen, Q. Liu, X. Zhong, A. Li, S. Liu, Y. Xiao, L. Chen, Y. Zhang, J. Su, T. Liu, R. Long, K. Shen, and L. Xiang, "Multi-SWE-bench: A Multilingual Benchmark for Issue Resolving," 2025. [Online]. Available: https://arxiv.org/abs/2504.02605

[72] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang, Y. Dai, J. Li, Z. Wang, R. Zhang, K. Wen, X. Ning, and Y. Wang, "FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGAs," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 223–234. [Online]. Available: https://doi.org/10.1145/3626202.3637562

[73] Z. Zeng, P. Chen, S. Liu, H. Jiang, and J. Jia, "MR-GSM8K: A Meta-Reasoning Benchmark for Large Language Model Evaluation," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=br4H61LOoI

[74] H. Zhang, A. Ning, R. B. Prabhakar, and D. Wentzlaff, "LLMCompass: Enabling Efficient Hardware Design for Large Language Model Inference," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 1080–1096.

[75] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasikci, "Atom: Low-bit quantization for efficient and accurate llm serving," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 196–209, 2024.

[76] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou, "Instruction-Following Evaluation for Large Language Models," 2023. [Online]. Available: https://arxiv.org/abs/2311.07911