Accepted: 22 September 2025
Revised: 07 September 2025
Received: 30 June 2025

RESEARCH-ARTICLE

CD-LLM: A Heterogeneous Multi-FPGA System for Batched Decoding of 70B+ LLMs using a Compute-Dedicated Architecture

WENHENG MA, Tsinghua University, Beijing, China

XINHAO YANG, Infinigence AI, Shanghai, China

SHULIN ZENG, Infinigence AI, Shanghai, China

TENGXUAN LIU, Infinigence AI, Shanghai, China

LIBO SHEN, Chinese University of Hong Kong, Hong Kong, Hong Kong

HONGYI WANG, Infinigence AI, Shanghai, China

View all

CD-LLM: A Heterogeneous Multi-FPGA System for Batched Decoding of 70B+ LLMs using a Compute-Dedicated Architecture

WENHENG MA*, Tsinghua University, China
XINHAO YANG*, Tsinghua University and Infinigence-AI, China
SHULIN ZENG, Tsinghua University and Infinigence-AI, China
TENGXUAN LIU, Tsinghua University and Infinigence-AI, China
LIBO SHEN, The Chinese University of Hong Kong, China
HONGYI WANG, Tsinghua University and Infinigence-AI, China
SHIYAO LI, Tsinghua University and Infinigence-AI, China
KE HONG, Tsinghua University and Infinigence-AI, China
ZHENHUA ZHU, Tsinghua University, China
XUEFEI NING, Tsinghua University, China
TSUNG-YI HO, The Chinese University of Hong Kong, China
GUOHAO DAI, Shanghai Jiao Tong University and Infinigence-AI, China
YU WANG, Tsinghua University, China

Large language models (LLMs) with 70 billion or more parameters are increasingly being deployed in cloud-based Model-as-a-Service (MaaS) scenarios. To meet the demands of such deployments, MaaS providers require batched LLM decoding systems that can deliver high system throughput (STP) while minimizing total cost of ownership (TCO). However, existing FPGA-based solutions predominantly focus on small-batch or single-batch inference, which fails to meet the computational requirements of batched LLM decoding, resulting in performance gaps of up to 7.96×. Moreover, the low utilization of multi-head attention operations in batched decoding scenarios, e.g., only 3.72% on A100 GPUs, further constrains throughput and inflates TCO.

To address these challenges, this paper introduces **CD-LLM**, a heterogeneous multi-FPGA system designed for efficient batched decoding of LLMs with 70B+ parameters, built upon a Compute-Dedicated architecture. First, we propose a memory-aligned mixed-precision quantization engine to reduce workload. By employing importance-aware quantization, we compress Llama-3.1-70B to an effective 3.45-bit representation and achieve 72.33% bandwidth utilization through memory-aligned

*Both authors contributed equally to this research.

Authors' Contact Information: Wenheng Ma, Tsinghua University, Beijing, China, wenhma@mail.tsinghua.edu.cn; Xinhao Yang, Tsinghua University and Infinigence-AI, Beijing, China, xh-yang24@mails.tsinghua.edu.cn; Shulin Zeng, Tsinghua University and Infinigence-AI, Beijing, China, zengshulin@mail.tsinghua.edu.cn; Tengxuan Liu, Tsinghua University and Infinigence-AI, Beijing, China, liutx21@mails.tsinghua.edu.cn; Libo Shen, The Chinese University of Hong Kong, Hong Kong, China, lbshen24@cse.cuhk.edu.hk; Hongyi Wang, Tsinghua University and Infinigence-AI, Beijing, China, wanghong22@mails.tsinghua.edu.cn; Shiyao Li, Tsinghua University and Infinigence-AI, Beijing, China, lishiyao20@mails.tsinghua.edu.cn; Ke Hong, Tsinghua University and Infinigence-AI, Beijing, China, hk24@mails.tsinghua.edu.cn; Zhenhua Zhu, Tsinghua University, Beijing, China, zhuzhenhua@mail.tsinghua.edu.cn; Xuefei Ning, Tsinghua University, Beijing, China, ningxuefei@mail.tsinghua.edu.cn; Tsung-Yi Ho, The Chinese University of Hong Kong, Hong Kong, China, tyho@cuhk.edu.hk; Guohao Dai, Shanghai Jiao Tong University and Infinigence-AI, Beijing, China, daiguohao@sjtu.edu.cn; Yu Wang, Tsinghua University, Beijing, China, yu-wang@tsinghua.edu.cn.

ACM 1936-7414/2025/10-ART
https://doi.org/10.1145/3771288

data packing. Second, we present a compute-dedicated FPGA architecture that maximizes peak performance by leveraging FPGA-specific resources such as DSPs, BRAMs, and LUTs. The compute-dedicated architecture enables CD-LLM to reach a peak performance of 59.90 TOPS at 600 MHz on U250 FPGA. At last, we introduce a heterogeneous master-slave multi-FPGA system to achieve higher utilization. By pipelining attention and linear layer computations across master and slave FPGAs, CD-LLM achieves utilization rates of 83.08% for linear layers and 68.30% for attention layers.

CD-LLM is designed with a heterogeneous multi-FPGA architecture, with an HBM-enabled FPGA as the master accelerator and eight DDR-based FPGAs as slave accelerators. When deployed for inference on the Llama-3.1-70B model with a batch size of 256, CD-LLM achieves a throughput of 2721.79 tokens/s. This represents a 6.11× improvement in STP and a 4.71× reduction in TCO compared to an eight-card RTX3090 GPU system. Furthermore, CD-LLM substantially outperforms the state-of-the-art eight-card FPGA accelerator FlightLLM, delivering 16.15× higher STP and 14.56× lower TCO.

CCS Concepts: • **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Additional Key Words and Phrases: LLM Accelerator, FPGA, Compute-Dedicated Architecture, Streaming Architecture

1 INTRODUCTION

Large language models (LLMs) have recently demonstrated impressive capabilities across various applications [1–3], driving their widespread adoption in cloud-based Model-as-a-Service (MaaS) scenarios. According to scaling laws [4], the size of LLMs has continued to increase to improve algorithmic accuracy [5, 6]. Modern MaaS systems now utilize LLMs with 70 billion or more parameters to achieve high accuracy [7, 8] and ensure the quality of generated responses, e.g., Meta Llama-3.1-70B/405B [9], xAI Grok-314B [10], Alibaba Qwen2-72B [11] and so on. The substantial storage and computing demands of 70B+ LLMs typically necessitate a multi-card inference system.

LLM inference consists of two stages: prefill and decode. The prefill stage processes all input tokens to generate the key-value cache (KV cache) and the initial output token, occurring only once per request. In contrast, the decode stage operates autoregressively to generate subsequent tokens using the KV cache and the previously generated token. Since the decode stage typically accounts for over 98% of the total inference latency [12], MaaS providers focus on optimizing batched decoding systems to maximize system throughput (STP) while minimizing total cost of ownership (TCO).

In modern cloud-based MaaS serving systems, batch sizes are typically set to around 256 to balance throughput and latency [13, 14]. For Llama-3.1-70B decoding at this batch size, the theoretical computation demand is 363.64 TOPS to offer sufficient system throughput, thereby ensuring per-user decoding speeds exceeding human reading rates [15] to ensure user experience. FPGA-based systems [16–29] have shown strong potential for low-latency LLMs decoding at small or single-batch. However, **current FPGA systems fall short in providing sufficient peak computing performance for batched LLM decoding**, as illustrated in Figure 1(a), exhibiting a significant 7.96× performance gap.

Quantization has proven to be highly effective in improving the efficiency of LLM inference. By compressing high-bit operations into lower-bit formats (e.g., FP16 to INT4), quantization reduces the computation workload of LLMs. However, existing low-bit quantization methods (e.g., below 4 bits) often result in significant accuracy loss. As shown in Figure 1(b), state-of-the-art AWQ [30] methods incur a 1.22% accuracy loss at 4-bit quantization, which increases to an unacceptable 8.54% at 3-bit quantization.

From a hardware perspective, existing FPGA designs often adopt control-centric or streaming architectures. While these approaches prove effective in application-specific integrated circuits (ASICs), they underperform on FPGAs when the computing demand is extremely high like batched 70B LLM decoding. This is because of insufficient consideration of FPGA-specific hardware resource layouts during architecture design, leading to suboptimal FPGA resource utilization and low peak computing performance. As Figure 1(c) shows, FlightLLM

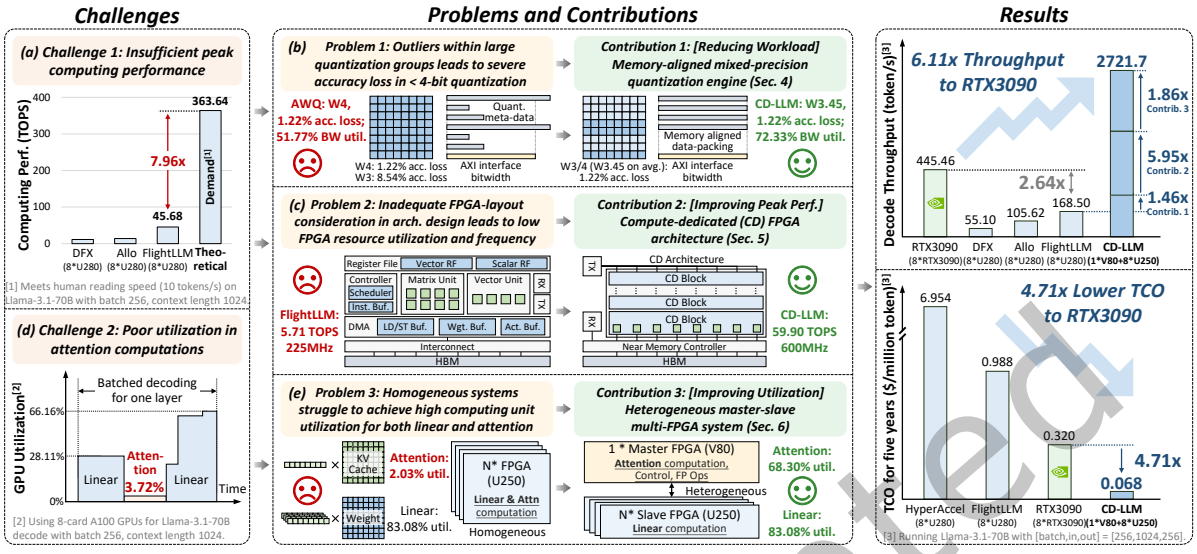

Fig. 1. Overview of the goals, challenges, problems, contributions and experiment results in CD-LLM.

operates at a frequency of only 225 MHz, delivering a peak performance of 5.71 TOPS, representing only 23.31% of the officially reported peak performance of the U280 FPGA [31].

Unfortunately, even with high-performance computing hardware, **the utilization of computing units for attention computation in LLM inference remains low**. For instance, in an eight-card A100 GPU system, attention layers achieve only 3.72% utilization in Llama-3.1-70B decoding with a batch size of 256 and a context length of 1024, accounting for approximately 18.65% of the total latency. In contrast, utilization rates for linear layers range from 28.11% to 66.16%, as illustrated in Figure 1(d). FPGA-based accelerators also face similar challenges with low utilization in attention, significantly limiting overall system performance and driving up TCO.

The attention utilization issue comes from that attention computations are memory-intensive, while linear computations are compute-intensive under large batch sizes. Therefore, homogeneous systems struggle to achieve high computing unit utilization for both linear and attention computations. As shown in Figure 1(e), a homogeneous AMD Alveo U250 multi-FPGA system achieves 83.08% utilization for linear layers but only 2.03% for attention layers.

To address these challenges, we propose **CD-LLM**, a heterogeneous multi-FPGA system using a Compute-Dedicated architecture, enabling 70B+ LLMs batched decoding. The key contributions are as follows.

- We propose a memory-aligned quantization engine to reduce workload. By incorporating importance-aware quantization and memory-aligned data packing, CD-LLM achieves 3.45-bit quantization with only a 1.22% accuracy loss on Llama-3.1-70B while attaining a 72.33% bandwidth utilization.
- We introduce a compute-dedicated FPGA architecture to improve peak performance. By incorporating the FPGA-specific layout of DSPs, BRAMs, and LUTs into architecture design, CD-LLM achieves a peak performance of 59.90 TOPS at 600 MHz on U250 FPGAs, representing a 10.49× improvement over the state-of-the-art FPGA accelerator FlightLLM, which delivers 5.71 TOPS at 225 MHz.

- We design a heterogeneous master-slave multi-FPGA system to get higher utilization. CD-LLM pipelines attention and linear computations on master and slave FPGAs, achieving computing unit utilization of 83.08% for linear layers and 68.30% for attention layers.

Implemented with one V80 FPGA as the master and eight U250 FPGAs as slaves, CD-LLM achieves a throughput of 2721.79 tokens/s on Llama-3.1-70B-Instruct with a batch size of 256, delivering 6.11× higher throughput and 4.71× lower TCO compared to eight-card RTX3090 system. Furthermore, CD-LLM reduces TCO by 3.70× compared to eight-card A100 GPU system. Compared to an eight-card FlightLLM system, CD-LLM delivers 16.15× higher STP and a 14.56× reduction in TCO.

2 BACKGROUND AND RELATED WORK

2.1 Background

2.1.1 Transformer-based LLMs. Currently, most transformer-based LLMs [9, 32] utilize a decoder-only architecture composed of multiple transformer [33] blocks. Each transformer block consists of two main components: the Multi-Head Attention (MHA) and the Feed Forward Network (FFN). During the inference process, input tokens are first embedded into a hidden space. The MHA then projects these embeddings into query Q, key K, and value V matrices, performing the attention mechanism for each head as described in Equation 1, where W_Q, W_K, W_V are the weight matrices for the projections. Subsequently, the FFN processes the attention output O through several fully connected layers along with a non-linear activation function f. The output of the FFN is then passed to the next transformer block.

$$Q = XW_Q, K = XW_K, V = XW_V, O = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{1}$$

The inference of LLMs can be divided into prefill and decode stage. These two stages exhibit significant differences in computational characteristics and resource requirements. During the prefill stage, all input tokens are processed in parallel. In contrast, the decode stage generates only one token at a time through an autoregressive process, making it highly bandwidth-sensitive and the primary contributor to inference latency. For instance, inference of Llama-33B on an NVIDIA A100 GPU, the latency ratio between prefill and decode is approximately 1:55, while the computational intensity ratio is about 110:1 [12]. Due to the differing demands of these two stages, separating them across multiple hardware platforms could result in higher throughput and efficiency [34–36]. In CD-LLM, we focus on accelerating batched LLM decoding on cloud FPGAs, while the implementation of the GPU-based prefill system is well explored and is not discussed in this paper.

2.1.2 Quantization. A vast amount of parameters are used in LLMs. To reduce storage and bandwidth overhead, quantization is one of the most effective methods to compress parameters to low-bit precision.

Most existing quantization methods partition weights into groups, typically of size 128, and quantize them to a uniform bitwidth. Within each group, the weights are quantized as Equation 2, where W represents the original FP16 weight, ZP and S denote the zero point and scaling factor shared in the group. The bitwidth of ZP typically matches that of the quantized weights, while S is commonly represented in FP16. As a result, smaller group sizes offer better data representation, leading to higher accuracy.

$$W_Q = \left\lfloor \frac{W - ZP}{S} \right\rceil, ZP = \min(W), S = \frac{\max(W) - \min(W)}{2^{(N_{bits}-1)} - 1} \tag{2}$$

Despite the compression in model size achieved, the challenges presented by the enormous size of current models cannot be fully addressed by the technique mentioned above. Even if all weights of a Llama-3.1-405B [37] model were compressed to INT4 format, it still needs more than 202.5 GB memory which beyond the capabilities of most accelerators, even the most advanced GPU (e.g., H200 with 141 GB [38]). Furthermore, during practical

inference, dynamic variables such as the KV cache and activations also require memory. With an increase in batched input length, the memory used for the KV cache may increase to 30% [39]. Consequently, it is necessary to utilize multiple devices, with each device storing a portion of the data.

2.2 Related Work

2.2.1 Model Compression. Quantization [40–42] reduces computational and memory overhead by converting weights and activations from high-bit to lower-bit precision. For better accuracy, recent quantization methods [30, 40] are activation-aware. They quantize the weights according to the activation distribution protecting the salient weights. Considering the varying importance of parameters, several works[30, 41–46] adopt mixed-precision quantization, using different data formats for weights and activations.

In addition to quantization, sparse computing is also used to reduce unnecessary computation and memory access. Some accelerators [27, 47] employ structured sparse computing units to skip zeros in weights, leading to better performance. However, structured sparse computing architectures need index data resulting in additional memory overhead. Alternatively, sparse attention [48–52] is proposed to reduce the length of KV cache. Since these technologies are all block-sparse compression, they can be used in different accelerators.

2.2.2 LLM-related Accelerators. Previous works [26, 27, 34–36, 51, 53–60] have introduced accelerators designed for transformer models. Some of these works [27, 51, 56, 59] focus on accelerating sparse attention and model inference on a single device. However, as the size of models grows, the memory capacity of a single accelerator becomes inadequate. DFX [26] and ExeGPT [60] address this limitation by distributing models across multiple devices. In addition, DFX particularly optimizes inference latency during the decode stage with efficient interconnections and DMA schemes.

However, these approaches still struggle with managing the interference and resource contention between the prefill and decode stages. More recent works like DistServe [34], Splitwise [36], and Mooncake [35], propose a disaggregated architecture that allocates prefill and decode operations to separate accelerators, enabling independent optimization for different design objectives (*e.g.*, TTFT and TPOT). Disaggregated architectures also face challenges related to low resource utilization in decode clusters, leading to substantial inefficiencies.

2.2.3 Streaming Architecture. Streaming architectures have been proposed in previous researches to optimize off-chip memory access [17, 18, 28, 61]. Data is streamed from PE to PE in these designs. FIFOs are inserted between PEs as the interface. Streaming architectures are friendly for high-level synthesis (HLS). But, they are usually not optimized for physical layout, causing poor resource utilization.

3 MOTIVATION AND ARCHITECTURE OVERVIEW

3.1 Motivation

In cloud-based MaaS scenarios, a batched decoding system is critical for higher STP and lower TCO. However, existing FPGA designs fall short of delivering adequate peak performance to meet the batched decoding computing demands. Additionally, attention computations suffer from low computing unit utilization. To address these challenges, guided by Equation 3, CD-LLM focuses on: *(1) reducing computational workload, (2) enhancing peak performance, and (3) improving computing unit utilization.*

$$Throughput = \frac{Peak\ Performance \times Utilization}{Computation\ Workload} \tag{3}$$

3.1.1 Mixed-precision Quantization to Reduce Workload. For specific FPGA-based designs, low-bit quantization can reduce computation workload by converting high-bitwidth operations into lower-bit operations. Existing coarse-grained quantization methods (e.g., with a group size of 128) perform well at 4-bit precision, reducing

the precision to 3-bit results in an unacceptable accuracy loss of 8.54%, as shown in Figure 1(b). The primary reason for this degradation is that **outliers within each quantization group cause the remaining data to be compressed to zero**. As described in Equation 2, these outliers lead to a very large scaling factor S for the group, which becomes even larger as the bitwidth N_{bits} decreases. A large S compresses smaller weight values W to zero, causing significant accuracy degradation. For example, there are 25.47% zeros in 3-bit quantized weights using AWQ where the group size of 128, while only 12.93% in 4-bit quantization. In addition, quantization introduces extra meta-data beyond weights, such as zero points and scaling factors. These meta-data are typically stored at separate memory addresses from the quantized weights within DRAM and on-chip buffers. Due to the nature of DRAM, non-contiguous fine-grain memory access reduces bandwidth utilization to 51.77% and limits the computing performance.

Therefore, smaller groups can mitigate the impact of outliers, enabling lower bit mixed-precision compression while maintaining accuracy. Smaller quantization groups are essential for implementing <4-bit mixed-precision quantization schemes, which further reduce computation workloads. However, due to the coarse-grained nature of single instruction multiple thread (SIMT) architectures, GPUs may not achieve performance gains from <4-bit mixed-precision quantization (e.g., 3/4-bit quantization) in smaller groups because of the additional dequantization overhead. In contrast, with custom architectural designs, FPGAs can effectively exploit lower bitwidths to achieve significant performance benefits.

3.1.2 Layout-aware Architecture Design to Improve Peak Performance. The layout and proportions of resources such as DSPs, BRAMs, and LUTs are fixed in FPGAs. This constraint forces designs on FPGAs to map onto existing cells and connect them via configurable wires rather than placed in optimal locations like in ASICs. Therefore, **current designs fail to align with the resource layout of FPGAs, resulting in suboptimal resource utilization**, longer interconnect wires, and poor timing, ultimately limiting peak performance, as illustrated in Figure 1(c). While this approach can meet less demanding computational needs, it leads to suboptimal resource utilization and timing issues. For example, DFX [26], Allo [28], and FlightLLM [27] operate at frequencies of 200 MHz, 245 MHz, and 225 MHz, delivering only 1.41 TFLOPS, 1.74 TOPS, and 5.71 TOPS, corresponding to just 5.76%, 7.10%, and 23.31% of the AMD Alveo U280 FPGA's peak performance [31]. These designs struggle to satisfy the computational requirements for batched decoding of 70B models, which demand up to 363.64 TOPS, leaving a performance gap of up to 7.96×.

The primary computational demand in LLM batched decoding comes from matrix multiplication. Inspired by the simple computation need of LLM batched decoding, designing a highly simplified compute-dedicated unit could improve FPGA resource utilization and peak performance. The compute-dedicated design adjusts the utilization of different resources according to the proportions guided by the physical layout, while simultaneously leveraging DSPs, BRAMs, and LUTs for MAC operations to enhance computational performance. Furthermore, the compute-dedicated unit separates on-chip buffers and control logic from the computing units. These optimizations improve resource utilization and operating frequency, ultimately boosting peak performance.

3.1.3 Heterogeneous System to Improve Utilization. **Batching has different effects on linear and attention layers due to their varying computational intensities**. As shown in Equation 4, where B represents the batch size, D_i and D_o denote the input and output dimensions, batching significantly increases the computational intensity (CI) for linear layers since all activations share the same weights. However, for attention layers, the KV cache data also scales with batch size, leaving the CI unchanged as the batch size grows. Consequently, attention computations are highly memory-bound, leading to low computing utilization. For instance, in an 8-card A100 GPU system, multi-head attention (MHA) achieves only 3.72% utilization of available computational performance. In contrast, linear layers, under large batch sizes, demand high-performance computing units to speedup, achieving utilizations ranging from 28.11% to 66.16% under the same configuration. Thus, homogeneous systems struggle to efficiently manage both attention and linear computations due to their contrasting resource

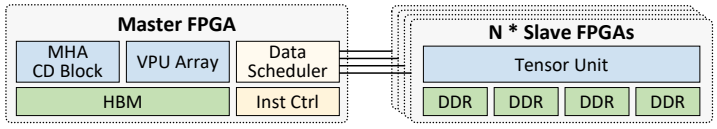

Fig. 2. The overall architecture of CD-LLM.

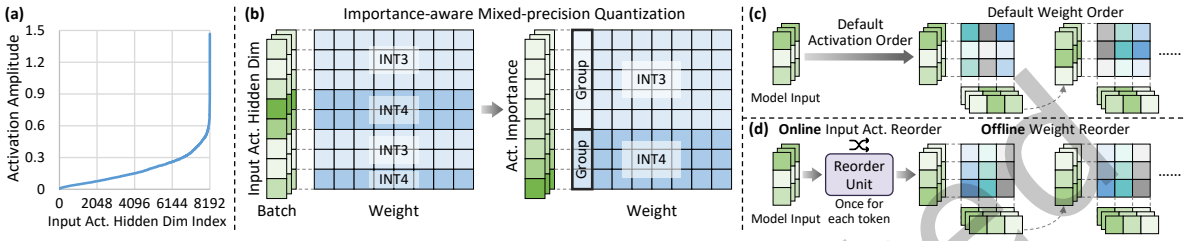

Fig. 3. (a) Sorted magnitudes of input activations in the 19th Q_{proj} layer of Llama-3.1-70B-Instruct on GSM8K. (b) Importance-aware mixed-precision quantization. (c) Inference without activation and weight reordering. (d) CD-LLM reorders weight columns/rows offline and performs a one-time input activation reorder online to adapt to the importance-aware mixed-precision quantization in (b).

requirements.

$$CI = \frac{B * D_i * D_o}{B * (D_i + D_o) + X * D_i * D_o}, X = \begin{cases} 1 & Linear, \\ B & Attention. \end{cases} \tag{4}$$

To address this challenge, a heterogeneous system can be utilized to handle attention and linear computations on different FPGAs. Attention computations can be offloaded to an HBM-equipped FPGA for higher memory bandwidth, while linear computations can be assigned to FPGAs optimized for high computational performance. Moreover, these two stages run on separate hardware and can be pipelined across requests to significantly enhance system throughput.

3.2 Architecture Overview of CD-LLM

As illustrated in Figure 2, CD-LLM is a heterogeneous multi-FPGA system comprising one master FPGA and N slave FPGAs. Each slave FPGA is equipped with two Tensor Units responsible for batched linear layer computations. The Tensor Units employ an FPGA-oriented compute-dedicated architecture, optimizing resource utilization and frequency, and achieving a performance of 59.90 TOPS. The master FPGA features an MHA CD Block, VPU Array, Data Scheduler, and Instruction Controller. The MHA CD Block also utilizes a compute-dedicated architecture, leveraging the HBM for attention computations. Additionally, the master FPGA performs all floating-point nonlinear operations, such as Softmax and SiLU. Detailed descriptions are provided in the following sections.

4 MIXED-PRECISION QUANTIZATION ENGINE

4.1 Importance-aware Quantization

Inspired by AWQ [30] and SmoothQuant [40], we have two key observations in model quantization. 1. Outliers can significantly affect the model's accuracy. This is because outliers bring large scales, making other values in the group be quantized to zero. 2. The distribution of the magnitude of the input activations introduces different importance of the corresponding weight rows, as shown in Figure 3(a). Based on these observations, we propose

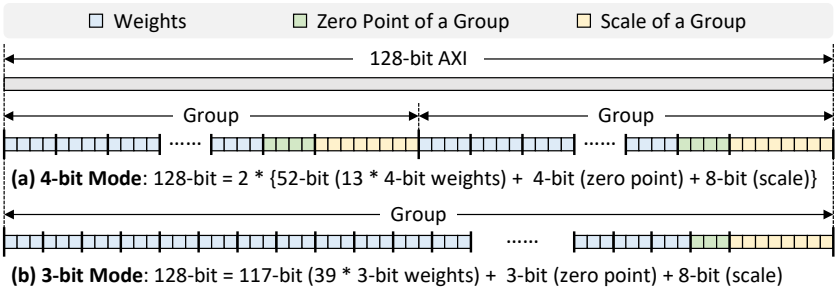

Fig. 4. Memory-aligned data packing for (a) 4-bit mode and (b) 3-bit mode. Each square represents a single bit of data.

the importance-aware mixed-precision quantization, assigning different bitwidths to weight rows based on their importance, to further exploit weight compression for better decoding performance.

During quantization, we first analyze the magnitude distribution of input activations for each linear layer. The weight rows are then reordered offline according to the magnitudes of their corresponding activations, from smallest to largest. Based on a predefined ratio $R\%$ of 3-bit weights, we select the first $R\%$ of rows with the smallest magnitudes and quantize them to 3-bit precision. Quantization is subsequently applied within each output hidden dimension, with groups spanning different input hidden dimensions, as shown in Figure 3(b). The quantized weight rows are stored in sorted order, placing 3-bit and 4-bit weights together to enable efficient access.

Next, the columns of each layer's weights are reordered offline according to the row order of the subsequent layer. This allows outputs from the previous layer to be directly used as inputs for the next layer, as illustrated in Figure 3(d). During online inference, a hardware reorder unit performs a one-time reorder of input activations to match the required order for the first linear layer, ensuring alignment with the sorted weight rows. Compared to inference without any offline weight reordering (Figure 3(c)), this approach requires only a single pre-processing step for input activations, which can be efficiently overlapped with model inference.

To maximize the fixed-point computing performance of FPGAs, we introduce a two-level scaling factor design in our quantization approach. Instead of using a single FP16 scaling factor per group, we decompose it into two levels: an INT8 scaling factor for each group and a shared FP16 scaling factor across all groups within the same column. The second-level INT8 scaling factor ensures that all MAC operations are performed using fixed-point arithmetic, avoiding inefficient floating-point computations that consume substantial hardware resources. Meanwhile, the first-level FP16 scaling factor retains sufficient data representation capability to preserve model accuracy. We select $R = 55\%$ in our quantization strategy, achieving 3.45-bit weight quantization with a maximum accuracy loss of 1.22% on Llama-3.1-70B-Instruct, as validated on the GSM8K and HumanEval datasets.

Notably, this importance-aware weight quantization scheme is a general approach applicable across different models. For example, AWQ [30] and MBQ [62] employ activation-magnitude-guided quantization strategies that leverage the importance of weights determined by activation magnitudes in LLMs and visual language models (VLMs), demonstrating the general applicability of this approach.

4.2 Memory-aligned Data Packing

During group-wise quantization, computations depend on various types of tensors, including low-bit activations, weights, and their associated zero points and scaling factors. Existing solutions demand multiple memory access to load these components into on-chip buffers before computation starts. However, the relatively small size of zero points and scaling factors (around 3.9% of total data for a group size of 128) results in fine-grained memory access, which significantly reduces effective bandwidth utilization to 51.77% in FlightLLM. To address this inefficiency,

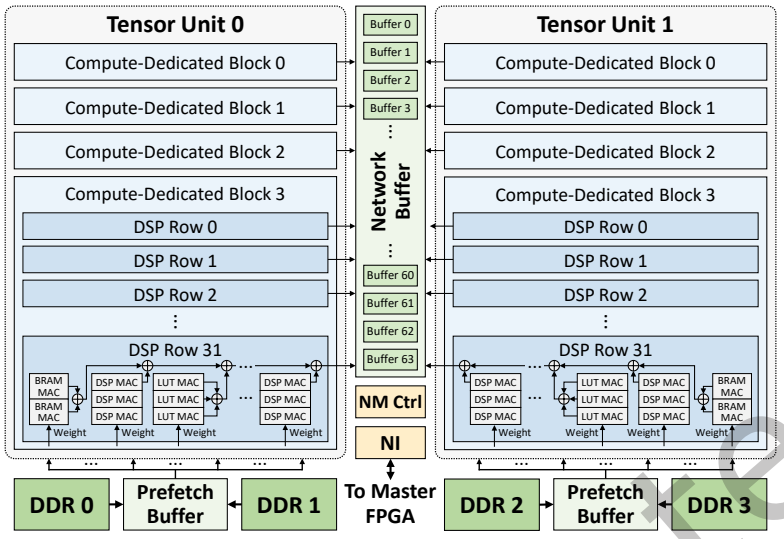

Fig. 5. The compute-dedicated architecture of slave FPGAs.

we propose a memory-aligned data-packing scheme, packing all necessary data for each computation together and aligning the bitwidth with the AXI interface during offline compilation. During inference, each data chunk aligns with the AXI interface bitwidth, enabling computations to commence immediately upon retrieval, reducing pipeline bubbles, and enhancing performance.

The memory-aligned data packing scheme is illustrated in Figure 4. For 4-bit mode, each 128-bit data line consists of two 64-bit groups. Each group contains 13 INT4 weights, one INT4 zero point, and one INT8 second-level scaling factor. For 3-bit mode, each 128-bit data line consists of one group, including 39 INT3 weights, one INT3 zero point, and one INT8 second-level scaling factor. This design ensures that each 128-bit data transaction that matches the bitwidth of the AXI interface contains all of the data for one or two quantization groups. In fact, smaller groups increase quantization meta-data. Considering the overhead of quantization meta-data, the average bitwidth of our approach is 3.94 bits, compared to 4.25 bits for AWQ's 4-bit method, which still demonstrates a lower overall bitwidth. Furthermore, our method achieves additional advantages in bandwidth utilization through memory-aligned data packing, further enhancing system efficiency.

Since the weights are grouped and stored as 3-bit and 4-bit during the quantization stage, we stored 3-bit and 4-bit weights in independent memory channels. Then, the weights and meta-data required for each computation are packed by the memory-aligned data packing scheme. This strategy enables offline precise allocation of specific computing units to each memory channel. By packing the grouped data to match the bitwidth of the AXI interface, we could maximize memory utilization and computational efficiency.

5 COMPUTE-DEDICATED FPGA ARCHITECTURE

5.1 Compute-dedicated Architecture for FPGA

5.1.1 Resource Mismatch in Previous FPGA-based Designs. As mentioned in 3.1.2, previous architecture designs always neglect the impact of the FPGA layout. For example, a streaming-based systolic array [63] [28], which is friendly for General Matrix Multiplication (GEMM) data flow, employs more on-chip buffer than DSP. We evaluate a 32×32 systolic array on VU3P and show the resource utilization in Table 1. Although the systolic

Table 1. 32 × 32 systolic array resource utilization on VU3P.

	LUT	BRAM (M18K)	DSP
VU3P	394k	1440	2280
Systolic Array	91.2k	1088	1024
Utilization	23.14%	75.56%	44.91%

array achieves high efficiency and scalability for GEMM operations, it fails to adapt to the resources on VU3P. Only 44.91% of DSPs are utilized consuming 75.56% of BRAMs and 23.14% LUTs. In this case, scaling is limited by insufficient BRAM resources wasting vast amount of LUTs. Besides, there are 17 DSP columns and 11 BRAM columns in VU3P. It is unfriendly to map a 32 × 32 systolic array on VU3P, leading to long wires and poor timing.

5.1.2 Compute-dedicated Architecture Designs. Data flow in LLM batched decoding is simple enough to detach all computing to a compute-dedicate block improving performance. As shown in Figure 5, slave FPGAs in CD-LLM all employ Compute-Dedicated (CD) Architecture making all resources focus on Tensor processing. Each slave FPGA contains two Tensor Units since U250 FPGA is split into two parts by the High-Performance IO. In each Tensor Unit, there are four Compute-Dedicated Blocks (CD Blocks) on each SLR for tensor computing. Each CD Block consists of 32 DSP Rows for 32 general matrix-vector multiplication (GEMV) operations. Slave FPGAs receive activations from the master FPGA via Network Interface (NI). All activations are preloaded to MAC units in DSP Row before tensor processing. The near-memory controller (NM Ctrl) will stream tensor data from the Prefetch Buffer to each DSP Row for GEMV operation. Accumulation results of all GEMV operations are kept in the network buffer for the next accumulation or transmission to the master FPGA.

5.1.3 Key Optimizations in CD Architecture. As illustrated in Figure 5, the CD architecture differs from previous compute- and control-centric designs in three key aspects:
Near-memory controller. Due to the simplified memory access pattern in LLM batched decoding, the complexity of the control unit can be greatly reduced. We move the controller near the DDR controller. It schedules data flow based on the status of the Prefetch Buffer and controls the target address for the network buffer.
Full streaming dataflow with simplified memory hierarchy. CD architecture employs a non-blocking streaming data flow for all data movements and computations. There are no stalls in data transfers or the computation pipeline in the CD Block, avoiding control or back-pressure logic in the Tensor Unit.
Computing-dedicated block. Since the CD Block is employed only for computing, all on-chip resources in the CD Block are accessible. All DSPs, BRAMs, and LUTs, according to the FPGA layout, are employed as MAC units to compose a hybrid CD Block, as shown in Figure 5. Due to the structured design, long wires are avoided in the CD block, leading to less congestion, higher clock frequency, and better resource utilization.

In contrast to conventional FPGA-based streaming architectures, the proposed CD architecture eliminates the need for input FIFOs and back-pressure control logic within individual processing units. It instead utilizes idle BRAMs and LUTs to perform MAC operations, thereby improving resource efficiency. This design achieves 59.90 TOPS on a Xilinx VU13P FPGA equipped with 12,288 DSPs, corresponding to 5.09 GOPS/DSP that substantially outperforms prior architectural approaches.

5.2 Dynamic BRAM-based MACs

In each CD block, all on-chip resources are devoted exclusively to computation. BRAMs, which are placed in columns adjacent to certain DSP columns on the FPGA, can be repurposed for MAC operations to boost peak

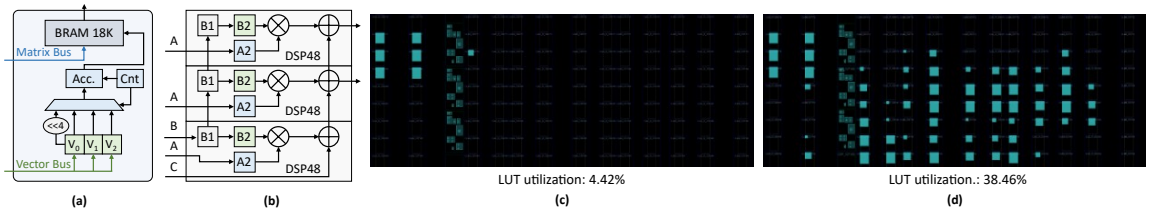

Fig. 6. (a) Using BRAMs for MACs. (b) The cascaded DSP structure. (c) Only using DSPs for MACs. (d) LUT-assisted DSPs for MACs.

Table 2. Frequency and LUTs utilization of different modules.

Module	**32b + 32b**	**4b × 8b**	**4b × 12b**	**8b × 8b**
Frequency	1124 MHz	813 MHz	778 MHz	710 MHz
LUT	32	31	43	73

performance. In CD-LLM, we employ both INT4- and INT3-quantized weights. Since each M18K memory block has a depth of 512 words, we map three INT3 MACs onto a single BRAM.

As shown in Figure 6(a), each BRAM-based MAC unit consists of a BRAM and a small update logic block. When a new vector configuration begins, three input vectors are sent to a vector accumulator. The accumulator's output, along with the address counter, drives the BRAM update logic. After the update, the BRAM holds all MAC results for those three activations across their respective weights. By accessing BRAM with input weights, the corresponding precomputed MAC result is retrieved.

The BRAM update procedure is detailed in Algorithm 1. During this routine, the element V_0 is added into the accumulator register *acc_result* eight times over nine consecutive cycles to generate the MAC results. On the fourth cycle, the accumulator subtracts $8 \times V_0$ to calibrate the *acc_result* for negative weights. When V_0 has been processed, V_1 is added to the *acc_result* once. And then V_0 updating is processed again. After V_1 is updated eight times, V_2 will be accumulated once. Similar to V_0 calibration, V_1 and V_2 will also be calibrated at the fourth accumulation. By streaming all three vector elements through a single adder, the design requires only one adder. Each BRAM holds the MAC lookup table for six elements in two vectors, achieving 12 OPs/cycle performance.

The BRAM update procedure requires 512 clock cycles. To mitigate the associated latency, a dual-BRAM architecture is employed in which one BRAM performs computation while the other undergoes updating. In typical LLM inference workloads, the number of weights per row frequently exceeds 2048, which provides sufficient computational duration to effectively conceal the update latency.

Unlike previous BRAM-based MACs, the MAC table in each BRAM is updated dynamically according to the input vector achieving better performance. Compared to customized BRAM-based MAC units (e.g., BRAMAC [64]), the existing basic BRAM block is enough for our dynamic BRAM-based MAC unit. In CD-LLM, 13 columns of BRAM are used for MACs, providing an additional 19.97 TOPs peak performance.

5.3 Resource Optimization

Increasing resource utilization on FPGA is often challenging due to insufficient wires. CD-LLM uses two types of in-DSP wire resources to minimize the need for additional wires: (1) POUT cascade resource (PCIN and PCOUT). Except for the last output in the DSP column, all DSP outputs are routed to the next DSP using the cascade path. (2) B cascade resource (BCIN and BCOUT). Vectors are placed at input port B, which has two input registers (B1

Algorithm 1 Dynamic BRAM-based MAC Table Update

Input: V_0, V_1, V_2
Output: $Mult\_LUT\_addr = addr\_cnt$,
$Mult\_LUT\_dat = acc\_result$

```
addr_cnt ← 0
acc_result ← 0
while addr_cnt < 512 do
  while addr_cnt[5 : 0] ≠ 0 do
    acc_result ← acc_result + V0
    addr_cnt ← addr_cnt + 1
    Update BRAM
    while addr_cnt[2 : 0] ≠ 0 do
      if addr_cnt[2 : 0] == 4 then
        acc_result ← acc_result − (V0 << 3)
      end if
      acc_result ← acc_result + V0
      addr_cnt ← addr_cnt + 1
      Update BRAM
    end while
    if addr_cnt[5 : 3] == 4 then
      acc_result ← acc_result − (V1 << 3)
    end if
    acc_result ← acc_result + V1
    Update BRAM
  end while
  if addr_cnt[8 : 6] == 4 then
    acc_result ← acc_result − (V2 << 3)
  end if
  acc_result ← acc_result + V2
  addr_cnt ← addr_cnt + 1
  Update BRAM
end while
```


and B2). Since vectors are rarely updated in LLM decoding, we cascade all B1 registers within a DSP column, as shown in Figure 6(b). New vector inputs are fed into the first B1 register and shift through the cascade path one by one. When new vectors are needed, an update signal is sent to all DSPs. This design reduces approximately 60% of input wires in DSP columns with 12-bit inputs on port B and 8-bit inputs on port A, enhancing overall DSP utilization.

Figure 6(c) shows 3 DSPs with 2 adders in a small block. In this block, only 4.42% LUTs are used. If we insert 6 multipliers and 4 adders in the block, as shown in Figure 6(d), 38.46% of LUTs are used to improve resource utilization. We evaluate different basic modules in Ultrascle+ FPGA, and the results are shown in Table 2. For basic modules, the logic could operate at 700 MHz, which is as fast as DSPs. According to the evaluation, idle LUTs are used as MAC units to achieve better peak performance.

Owing to the complex memory hierarchy and control mechanisms, previous FPGA designs typically run at lower frequencies, around 200+ MHz. However, this architecture enables full pipelining of all logic circuits without

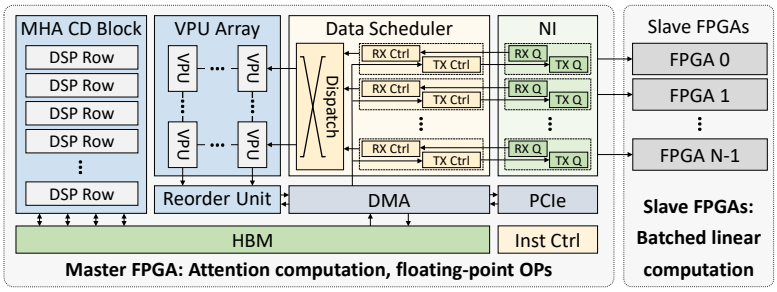

Fig. 7. Micro-architecture of the master FPGA.

any blocking or back-pressure. Thus, the CD Block could work at a much higher frequency (600 MHz), including BRAMs, LUTs, and DSPs. Other parts work at 300 MHz. This approach eliminates the overhead associated with DSP frequency doubling and reduces the need for additional input or output multiplexers (MUXes). Considering all the above mentioned optimizations, DSPs, BRAMs and LUTs provide 28.57 TOPS, 19.97 TOPS and 11.36 TOPS computing performance, respectively. Finally, CD-LLM achieves 59.90 TOPS on each U250 FPGA.

6 HETEROGENEOUS MULTI-FPGA SYSTEM

6.1 Master-Slave Architecture

6.1.1 System Architecture. As illustrated in Figure 7, the multi-FPGA system is structured using a master-slave architecture with a star topology, where all slave FPGAs are connected to the master FPGA via an Aurora-based [65] network interface. The Vector Unit on the master V80 FPGA communicates with Tensor Units across eight U250 FPGAs through the Network Interface (NI). Both the Tensor Units on slave FPGAs and the MHA Compute-Dedicated Block (MHA CD Block) on the master FPGA are designed with the compute-dedicated architecture to deliver high performance for batched decoding. The Instruction Controller manages the scheduling among the Vector Unit, the MHA CD Block, and the Tensor Units across FPGAs, issuing commands to initiate computations or memory access as required. Additionally, a Reorder Unit is included to perform online reordering of input activations based on the row indices of the first weight matrix, enabling mixed-precision computations.

6.1.2 Optimization of Linear and Attention. Due to the distinct computational characteristics of linear and attention layers in LLM batched decoding, attention computations demand high memory bandwidth, while batched linear computations primarily require high computational performance.

To address these differences, CD-LLM employs a heterogeneous system that separates attention and linear computations across different hardware platforms and establishes a pipeline between them. The master node, a V80 FPGA equipped with HBM, is optimized for bandwidth-intensive attention computations. The slave nodes, U250 FPGAs, leverage a large number of DSPs at a low cost, providing high computational performance and lower TCO. This design enables request-level computation parallelism and maximizes hardware utilization.

Attention computations (QK^T, Softmax, PV) are executed on the master V80 FPGA, while linear computations are assigned to the slave U250 FPGAs. During inference, a task scheduler on the master FPGA coordinates pipelined execution and communication between the master and slave FPGAs. To maximize overlap and utilization, we evenly split batched decoding into two micro-batches: while the master processes the attention of the first micro-batch, the slave FPGAs concurrently perform the linear computations of the second. This alternating execution continues across micro-batches, minimizing idle time caused by data dependencies between attention and linear computations. As a result, the system achieves an end-to-end performance improvement of 1.12×–1.86×, with

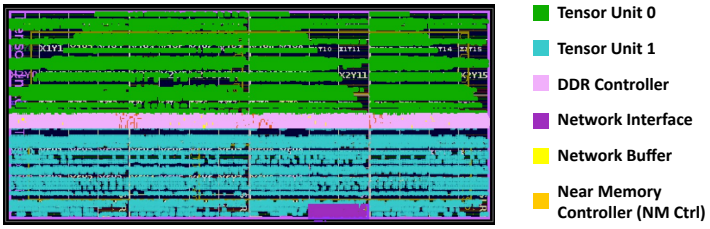

Fig. 8. Layout on Slave FPGA (U250).

computing utilization reaching 83.08% for linear computations on the slave FPGAs and 68.30% for attention computations on the master FPGA.

6.1.3 Discussion. In the current system configuration, the primary cause of idle compute units is the network transmission latency. This limitation arises because the U250 board is equipped with only two network ports, although the FPGA chip itself supports additional network interface expansions. Notably, when processing larger models, the relative workload on network transmission reduces because of longer computing latency, resulting in higher utilization.

7 EVALUATION

7.1 Evaluation Setup

Models and Metrics. We choose the state-of-the-art LLM Llama-3.1-70B-Instruct [9] to evaluate our design. We evaluate the accuracy of the original FP16 model, weight-only quantized with AWQ [30], and our proposed mixed precision quantization under the commonly used GSM8K [66] and HumanEval [67] datasets. We select GSM8K to evaluate mathematical reasoning and chain-of-thought capability, and HumanEval to assess code generation and functional correctness. All evaluations are performed on the full test subsets, i.e., 1319 problems for GSM8K and 164 problems for HumanEval.

The STP of the decode stage is defined as the number of output tokens divided by the duration of the decode stage (token/s). Note that CD-LLM only accelerates the LLM batched decoding for disaggregated prefill/decode systems. Therefore, prefill operations are not considered in the CD-LLM system. We use tensor parallelism on eight cards for both GPU and FPGA systems in all evaluations. All GPU and FPGA experiments are repeated five times. The performance is very stable, with a standard deviation of less than 1% on GPUs and less than 0.3% on FPGAs, so we report the average performance in our evaluation.

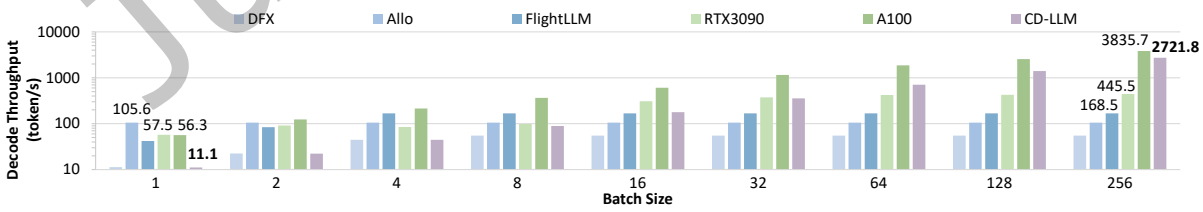

Fig. 9. Decode throughput under different batch size with [input, output] = [1024, 256].

Table 3. System configurations for the baselines and CD-LLM, with hardware parameters reported for a single card.

System	Hardware	Card Num	Freq. (MHz)	Perf. (TOPS)	Mem. (GB)	BW (GB/s)	Power (W)
DFX [26]	Alveo U280 (16nm)	8	200	1.41	16	460	45
Allo [28]	Alveo U280 (16nm)	8	245	1.74	16	460	30
FlightLLM [27]	Alveo U280 (16nm)	8	225	5.71	16	460	45
HyperAccel [68]	Alveo U55C (16nm)	8	(Not Reported)		16	460	75
RTX3090 GPU	RTX3090 (8nm)	8	1695	284	24	936	209
A100 GPU	A100-SXM4-80GB (7nm)	8	1410	624	80	2040	320
CD-LLM (Master)	Alveo V80 (7nm)	1	300	13.98	32	820	140
CD-LLM (Slave)	Alveo U250 (16nm)	8	600	59.90	64	77	190

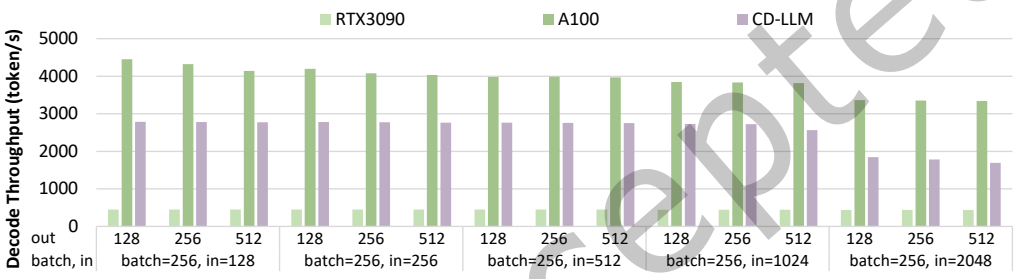

Fig. 10. Decode throughput of CD-LLM and GPU system under different [input, output] with batch size 256.

GPU Baselines. We compare CD-LLM against both GPU and state-of-the-art FPGA systems, as detailed in Table 3. For GPU systems, we utilize the TensorRT-LLM [69] framework with W4KV8[1] quantization, running on a server with eight A100-SXM4-80GB GPUs and another server with eight RTX3090 GPUs. GPU power consumption is measured using NVIDIA's system management interface (`nvidia-smi`).

FPGA Baselines. For FPGA LLM accelerators, we include DFX, FlightLLM, Allo, and HyperAccel in our comparison. We summarize below the methods used to evaluate the performance and power of different FPGA LLM baselines.

- **DFX and FlightLLM:** Performance is measured using FlightLLM's performance simulator [70] for a single U280 card on a 7B model, under the corresponding input-output sequence lengths and batch sizes. The results are then extrapolated to a 70B model based on actual computational and memory requirements, assuming ideal linear scaling across eight cards. Power consumption values are directly taken from the respective publications [26, 27].
- **Allo:** Performance for the 70B model is estimated from the reported Vicuna-13B inference results [28], again assuming linear scaling across eight cards. Power consumption is conservatively estimated at 30W due to the lack of reported data.
- **HyperAccel:** Performance of the OPT-66B model under an eight-card FPGA setup is taken directly from the official website [68] and its datasheet [71]. Power consumption is also sourced from the datasheet.

[1]WxKVy denotes x-bit quantization for weights and y-bit quantization for KV caches.

Table 4. Resource utilization on Master (V80) and Slave (U250) FPGAs

Utilization	Master (V80)	Slave (U250)
LUT	971k (37.72%)	698k (40.39%)
FF	1580k (30.69%)	1680k (48.61%)
BRAM	1275 (34.33%)	1776 (66.07%)
URAM	960 (49.87%)	80 (6.25%)
DSP	6108 (56.31%)	11356 (92.42%)

In the multi-FPGA scalability experiments (Table 7), we directly use the data reported from the DFX paper [26] and the HyperAccel website [68]. Note that the 8-card scalability for DFX is not reported.

FPGA Implementation.We implement CD-LLM in Verilog using Vivado 2024.1. Figure 8 illustrates the layout of CD-LLM on the U250 FPGA. And Table 4 shows the resource utilization for on the Master and Slave FPGAs. Power consumption is measured through on-board sensors.

7.2 Evaluation Results

7.2.1 Inference Accuracy. As outlined in Equation 2, the quantized weight within a group is determined by the extremal values (the maximum and minimum weights). When outliers, defined as weights significantly larger or smaller than the majority, are present in a quantization group, they disproportionately influence the quantization range. This results in a greater number of weights near zero being quantized to zero, thereby exacerbating accuracy degradation. In our proposed method, this issue is effectively alleviated by smaller quantization groups, which reduce the influence of outliers. As a result, in the AWQ approach with a group size of 128, the proportion of weights quantized to zero reaches 12.93% for 3-bit quantization and 25.47% for 4-bit quantization. In contrast, our 3.45-bit quantization strategy results in a zero ratio of just 13.13%, closely matching the accuracy of AWQ's 4-bit quantization while using fewer bits.

CD-LLM employs an importance-aware mixed-precision quantization strategy, which facilitates more aggressive weight compression without compromising model accuracy. Empirical evaluations conducted on the GSM8K and HumanEval benchmarks substantiate the efficacy of our approach. As presented in Table 5, our mixed-precision quantization incurs a maximum accuracy loss of only 1.22%, which is comparable to the 4-bit AWQ quantization despite utilizing lower bitwidths for weights and KV cache. Notably, CD-LLM demonstrates superior accuracy relative to the 3-bit AWQ method, owing to its selective compression mechanism: only less significant weights are quantized to 3-bit, while preserving critical weights at 4-bit precision.

However, the coarse-grained feature of the SIMT architecture makes it challenging for GPUs to gain additional performance from fine-grained mixed-precision quantization. FPGA-based CD-LLM, on the other hand, may gain more effectively from fine-grained mixed-precision quantization in terms of reduced access memory and improved DSP efficiency. In this scenario, the four DDR channels on the U250 are evenly divided between 3-bit and 4-bit weights.

The inherently coarse-grained execution model of GPU architectures limits their ability to fully exploit the performance benefits of fine-grained mixed-precision quantization. In contrast, FPGA-based CD-LLM is better suited to leverage such granularity, offering tangible advantages in terms of reduced memory access overhead and enhanced DSP utilization. In our design, the four DDR memory channels on the Xilinx U250 FPGA are evenly partitioned to store 3-bit and 4-bit weights, thereby enabling balanced bandwidth utilization and optimized computational throughput.

Table 5. Accuracy loss under different quantization methods.

Quantization Method	**GSM8K** [66]	**HumanEval** [67]
Original Model (FP16)	91.96 (-)	46.34 (-)
AWQ [30] (W4KV8)	91.89 (-0.07)	45.12 (-1.22)
AWQ [30] (W3KV8)	90.07 (-1.89)	37.80 (-8.54)
CD-LLM (W3.45KV4)	**91.51 (-0.45)**	**45.12 (-1.22)**

Table 6. Efficiency of DSPs (for FPGAs) and Tensor Cores (TCs, for GPUs), along with power efficiency across different systems. GPU Tensor Core performance is evaluated using INT8.

Platform	**Hardware**	**#DSP/#TC in System**	**STP (token/s)**	**Power (W)**	**DSP/TC Eff. (GOPS per Unit)**	**Energy Eff. (token/J)**
RTX3090	GPU	2624	445.46	1673	865.85	0.27
A100	GPU	3456	3835.66	2564	1444.44	1.50
Allo [63]	FPGA	14.24k	105.62	240	0.98	0.44
FlightLLM [27]	FPGA	49.15k	168.50	360	0.93	0.47
CD-LLM	FPGA	96.96k	2721.79	1310	5.09	2.08

7.2.2 STP under Different Batch Sizes. Figure 9 presents a comparative analysis of the performance of CD-LLM against baseline systems across varying batch sizes. The evaluation uses input and output token lengths of [1024, 256], consistent with the average sequence lengths observed in the widely adopted ShareGPT [72] dataset.

At smaller batch sizes (e.g., batch size ≤ 8), CD-LLM exhibits relatively lower performance. This is primarily due to its use of U250 FPGAs without HBM as slave devices, a design choice that lowers system cost but also restricts off-chip memory bandwidth. Since small-batch decoding is highly memory-intensive, this limitation reduces throughput.

Conversely, at larger batch sizes (e.g., batch size ≥ 32), CD-LLM demonstrates superior performance, benefiting from its substantially higher peak computational capacity compared to other FPGA-based systems. In typical MaaS cloud deployment scenarios, where batch sizes commonly reach 256 [13, 14], CD-LLM achieves a throughput of 2721.79 tokens per second. This performance corresponds to a speedup of 16.15× over an 8-card FlightLLM system, 6.11× over an eight-card RTX3090 GPU system, and 0.71× relative to an eight-card A100 GPU system.

7.2.3 STP under Different Input and Output. We assess the performance of CD-LLM across a range of input-output sequence lengths under diverse evaluation scenarios. Following prior work [34] on profiling real-world workload datasets including ShareGPT [72], HumanEval [67], and LongBench [73], we select representative context and output lengths based on the distribution observed in practical datasets, covering context lengths from 128 to 2048 tokens and output lengths from 128 to 512 tokens.

As shown in Figure 10, when the context length increases, the share of memory-bound attention computations grows, causing the performance of CD-LLM to be bottlenecked by the bandwidth of the master node. Even under this constraint, CD-LLM achieves an average speedup of 5.72× relative to the RTX3090 system, while delivering approximately 65% of the throughput attained by the A100 system. These results highlight the competitive efficiency of CD-LLM, particularly in resource-constrained or cost-sensitive deployment environments.

7.2.4 DSP and Power Efficiency. CD-LLM achieves a peak performance of 59.90 TOPS, outperforming the FlightLLM and Allo systems by factors of 10.49× and 34.43×, respectively. As detailed in Table 6, CD-LLM also

Table 7. Scalability of multi-FPGA systems, with the number of FPGAs indicating the count of slave FPGAs in CD-LLM.

System	1 FPGA	2 FPGA	4 FPGA	8 FPGA
DFX [26]	1.00×	1.57×	2.23×	(Not Reported)
HyperAccel [68]	1.00×	1.80×	3.13×	5.39×
CD-LLM	**1.00**×	**1.99**×	**3.91**×	**7.62**×

Table 8. Five-year TCO across different systems.

System	Model	STP (token/s)	Purchase Cost ($)	Energy Cost ($)	TCO ($/M token)
HyperAccel [68]	OPT-66B	23.60	24k	3754.29	7.458
FlightLLM [27]	Llama-3.1-70B	168.50	24k	2252.57	0.988
RTX3090 GPU	Llama-3.1-70B	445.46	12k	10466.69	0.320
A100 GPU	Llama-3.1-70B	3835.66	136k	16042.12	0.251
CD-LLM (V80+U250)	Llama-3.1-70B	2721.79	95k	8196.86	**0.240**
CD-LLM (VH1872+VU13P)	Llama-3.1-70B	2721.79	21k	8196.86	**0.068**
CD-LLM (VH1872+VU13P)	Llama-3.1-405B	479.42	21k	8196.86	**0.386**

demonstrates substantial improvements in resource utilization. Specifically, CD-LLM enhances DSP efficiency by 5.19× over Allo and 5.47× over FlightLLM. These gains are primarily attributed to BRAM-based MAC and the compute-dedicated architecture, which together enable more efficient exploitation of FPGA resources.

In terms of energy efficiency, CD-LLM delivers 4.73× and 4.42× improvements over Allo and FlightLLM, respectively, as shown in Table 6. When compared to mainstream GPU platforms, it achieves 7.81× greater energy efficiency than the RTX3090 system and 1.39× greater than the A100 system, highlighting its suitability for high-performance, energy-efficient deployment scenarios.

7.2.5 Ablation Study. The performance improvements observed in CD-LLM can be attributed to three key architectural and algorithmic enhancements, as illustrated in Figure 1. First, the use of lower bit quantization significantly reduces computational overhead, contributing a 1.46× speedup. Second, the adoption of a compute-dedicated architecture yields a substantial increase in peak computational throughput, resulting in a 5.95× improvement. Third, the deployment of a heterogeneous multi-FPGA system enhances overall resource utilization, delivering an additional 1.86× performance gain. Collectively, these factors underpin the efficiency of CD-LLM in high-throughput inference scenarios.

7.2.6 Scalability. As shown in Table 7, CD-LLM demonstrates superior scalability compared to existing multi-FPGA designs as the number of FPGAs increases. With eight slave FPGAs, CD-LLM achieves a 7.62× performance improvement relative to a single slave FPGA. This scalability is primarily enabled by its master-slave architecture, which effectively orchestrates aggregation-intensive operations commonly found in LLM workloads, such as LayerNorm and Softmax.

Moreover, while inter-FPGA communication latency introduces some performance overhead, the system design mitigates this through efficient pipelining. In particular, larger batch sizes help to amortize non-computational latency, thereby improving overall throughput and maintaining high utilization across the FPGA array.

[1]The data for DFX [26] and HyperAccel [68] are sourced from their paper or website.

7.2.7 Inference Cost. Table 8 presents a comprehensive TCO analysis comparing CD-LLM with several baseline systems. The TCO, expressed in dollars per million generated tokens, is computed as the sum of purchase and energy costs over five years [74], as shown in Equation 5. Here, C_{purchase} denotes the hardware purchase cost over five years, $Power$ is the average system power consumption (W), T denotes the time over five years, γ represents the electricity price (\$/kWh), and N_{token} is the total number of tokens generated within five years. The TCO is expressed in \$/million token. We selected $\gamma = 0.14$ \$/kWh based on the average U.S. electricity price [75].

$$\text{TCO} = \frac{C_{\text{purchase}} + Power \times T \times \gamma}{N_{\text{token}}/10^6} \tag{5}$$

For CD-LLM, two configurations are considered: (1) *CD-LLM (V80+U250)*, which utilizes AMD Alveo accelerator cards, and (2) *CD-LLM (VH1872+VU13P)*, which employs in-house FPGA cards. The cost estimates for in-house FPGA configurations, as well as for other FPGA-based systems such as FlightLLM and HyperAccel, include both the FPGA chip and associated board components. Pricing data for FPGA chips is sourced directly from vendors [76] and reflects bulk procurement volumes typical of cloud-based MaaS providers (purchase quantity: 5,000–10,000).

Compared to GPU-based systems, CD-LLM provides a substantial advantage in TCO, a critical metric for cloud-scale deployment. Although its performance is marginally lower than that of the A100, CD-LLM achieves significantly lower per-token cost due to its reduced hardware and energy expenses. As shown in Table 8, CD-LLM delivers a 4.71× and 3.70× reduction in TCO for 70B model inference relative to RTX3090 and A100 systems, respectively. Furthermore, CD-LLM can support 405B model inference at a comparable TCO, whereas the GPU systems are limited to 70B inference under similar cost constraints. This advantage stems from CD-LLM 's competitive performance, combined with its markedly lower acquisition cost and power consumption. Compared to other FPGA-based solutions, CD-LLM achieves over 14× better TCO, primarily due to its superior resource utilization.

8 DISCUSSION

The preceding sections have detailed the design of CD-LLM and demonstrated its high performance and efficiency. However, its achievable throughput is ultimately bounded by the hardware constraints of contemporary FPGA platforms. In this section, we discuss the impact of Mixture-of-Experts (MoE) models and explore prospective developments in FPGA technology for LLM inference.

Limitations of Existing FPGAs. As demonstrated in the evaluation results, CD-LLM delivers a superior TCO compared to the NVIDIA A100 GPU, albeit with lower end-to-end throughput. This performance gap is primarily attributable to limited peak computational capability. Leveraging the CD architecture, CD-LLM attains 59.90 TOPS on a Xilinx VU13P FPGA, still less than one-twentieth of the peak performance of an A100 GPU. The VU13P, introduced in 2016, employs DSPs for MAC operations and DDR4-2400 memory, both of which impose significant constraints on large-scale LLM inference. Furthermore, most of the device's SerDes remain unused in the current CD-LLM implementation, despite their potential to provide higher network bandwidth. These observations suggest that CD-LLM could achieve substantially greater throughput if deployed on platforms with increased computational resources and higher memory bandwidth.

Impact of MoE Models. MoE-based models exhibit significantly lower computational intensity compared to dense models. For instance, the DeepSeek-R1 [77] comprises 256 experts, of which only 8 are activated during each inference, corresponding to just $\frac{1}{32}(\frac{8}{256})$ of the operations required by a dense model. This reduced demand for peak performance makes MoE models well-suited to FPGA-based systems. Moreover, large-scale MoE models typically require greater memory capacity. For DDR-based FPGA platforms, this characteristic can facilitate improved TCO.

Table 9. CD-LLM with hypothetical hardware configurations.

CD-LLM Configuration	Master Bandwidth (GB/s)	Slave Performance (TOPS)	Slave Memory	Slave Bandwidth (GB/s)
V80+U250	820 (V80)	59.90	4×DDR4-2400	79.2
Config A	2040 (same as A100)	120	4×DDR5-4800	153.6
Config B	3350 (same as H100)	240	6×DDR5-6400	307.2
Config C	4800 (same as H200)	360	8×DDR5X-8533	546.1

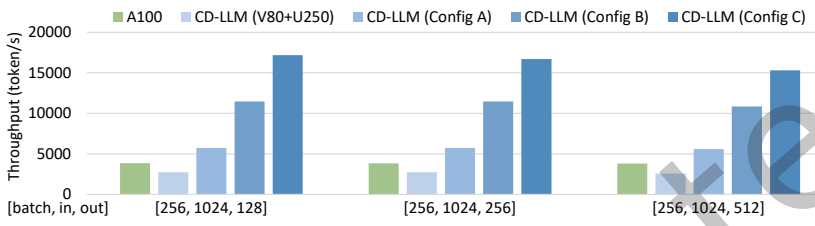

Fig. 11. Decode throughput of an eight A100 GPU system and CD-LLM under different hardware setups.

Outlook for Future FPGAs. For LLM inference, both higher computational performance and greater memory bandwidth will be essential. We evaluate the projected performance of CD-LLM under several hypothetical hardware configurations, as summarized in Table 9. In these projections, the memory bandwidth of the master FPGA is assumed to match that of various GPUs, reflecting the memory-bound nature of attention operations. The performance of the slave FPGAs is assumed to align with the MX6 performance of AMD Versal AI Edge Series Gen 2 devices [78] equipped with different off-chip memory types.

As illustrated in Figure 11, a configuration with four DDR5-4800 memory modules and 120 TOPS per slave FPGA yields a 1.5× throughput improvement, despite each slave FPGA delivering only one-tenth of the peak performance of an NVIDIA A100 GPU. Owing to their lower TCO, FPGAs have the potential to provide superior performance in future LLM inference workloads, particularly for low-operation-intensity tasks such as attention and MoE processing. Relative to the baseline CD-LLM system, enhanced hardware configurations achieve throughput gains of 2.13×, 4.21×, and 6.14×, respectively. These results indicate that, with anticipated advances in FPGA performance and memory bandwidth, substantial improvements in FPGA-based LLM inference systems are achievable.

9 CONCLUSION

In this paper, we introduce CD-LLM, a heterogenous multi-FPGA system, enabling cloud FPGAs available for batched decoding of 70B+ LLMs. CD-LLM apply memory-aligned quantization engine, compute-dedicated architecture design and heterogeneous master-slave mutli-FPGA system techniques. CD-LLM achieves a throughput of 2721.79 token/s on Llama-3.1-70B. Compared to eight-card RTX3090 GPU system, CD-LLM delivers a 6.11× increase in STP and a 4.71× reduction in TCO for 70B LLM decoding.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (2023YFB4502200), the National Natural Science Foundation of China (No. 62325405, 62104128, 62203257, 62031017, 62406159, U21B2031), Tsinghua University Initiative Scientific Research Program, Tsinghua EE Xilinx AI Research Fund, Tsinghua-Meituan Joint

Institute for Digital Life. Tsinghua-Efort Joint Research Center for EAI Computation and Perception, Beijing National Research Center for Information Science, Technology (No. BNR2024TD03001), Beijing Innovation Center for Future Chips, and State Key laboratory of Space Network and Communications.

REFERENCES

[1] Siyuan Chen, Mengyue Wu, Kenny Q Zhu, Kunyao Lan, Zhiling Zhang, and Lyuchun Cui. Llm-empowered chatbots for psychiatrist and patient simulation: Application and evaluation. arXiv preprint arXiv:2305.13614, 2023.
[2] Chaozheng Wang, Junhao Hu, Cuiyun Gao, Yu Jin, Tao Xie, Hailiang Huang, Zhenyu Lei, and Yuetang Deng. Practitioners' expectations on code completion. ArXiv, abs/2301.03846, 2023.
[3] Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. Large language models in medicine. Nature medicine, 29(8):1930–1940, 2023.
[4] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361, 2020.
[5] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. arXiv preprint arXiv:2303.18223, 2023.
[6] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey. arXiv preprint arXiv:2402.06196, 2024.
[7] Together.AI. Together.ai products: Inference, 2024. Accessed: October 8, 2024.
[8] Amazon Web Services. Amazon bedrock: Foundation models at scale, 2024. Accessed: October 8, 2024.
[9] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.
[10] xAI. Announcing grok-1.5, 2024.
[11] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. arXiv preprint arXiv:2407.10671, 2024.
[12] Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-thought: Large language models can do parallel decoding. Proceedings ENLSP-III, 2023.
[13] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Proceedings of the 29th Symposium on Operating Systems Principles, pages 611–626, 2023.
[14] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. arXiv preprint arXiv:2308.16369, 2023.
[15] Hanchen Li, Yuhan Liu, Yihua Cheng, Siddhant Ray, Kuntai Du, and Junchen Jiang. Eloquent: A more robust transmission scheme for llm token streaming. In Proceedings of the 2024 SIGCOMM Workshop on Networks for AI Computing, pages 34–40, 2024.
[16] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pages 75–84, 2017.
[17] Linghao Song, Yuze Chi, Atefeh Sohrabizadeh, Young-kyu Choi, Jason Lau, and Jason Cong. Sextans: A streaming accelerator for general-purpose sparse-matrix dense-matrix multiplication. In Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '22, page 65–77, New York, NY, USA, 2022. Association for Computing Machinery.
[18] Linghao Song, Yuze Chi, Licheng Guo, and Jason Cong. Serpens: A high bandwidth memory based accelerator for general-purpose sparse matrix-vector multiplication. In Proceedings of the 59th ACM/IEEE design automation conference, pages 211–216, 2022.
[19] Yixiao Du, Yuwei Hu, Zhongchun Zhou, and Zhiru Zhang. High-performance sparse linear algebra on hbm-equipped fpgas using hls: A case study on spmv. In Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pages 54–64, 2022.
[20] Abhishek Kumar Jain, Chirag Ravishankar, Hossein Omidian, Sharan Kumar, Maithilee Kulkarni, Aashish Tripathi, and Dinesh Gaitonde. Modular and lean architecture with elasticity for sparse matrix vector multiplication on fpgas. In 2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pages 133–143. IEEE, 2023.
[21] Manoj B Rajashekar, Xingyu Tian, and Zhenman Fang. Hispmv: Hybrid row distribution and vector buffering for imbalanced spmv acceleration on fpgas. In Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pages 154–164, 2024.
[22] Zhengang Li, Mengshu Sun, Alec Lu, Haoyu Ma, Geng Yuan, Yanyue Xie, Hao Tang, Yanyu Li, Miriam Leeser, Zhangyang Wang, et al. Auto-vit-acc: An fpga-aware automatic acceleration framework for vision transformer with mixed-scheme quantization. In 2022 32nd International Conference on Field-Programmable Logic and Applications (FPL), pages 109–116. IEEE, 2022.

[23] Peiyan Dong, Mengshu Sun, Alec Lu, Yanyue Xie, Kenneth Liu, Zhenglun Kong, Xin Meng, Zhengang Li, Xue Lin, Zhenman Fang, et al. Heatvit: Hardware-efficient adaptive token pruning for vision transformers. In 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pages 442–455. IEEE, 2023.
[24] Jinming Zhuang, Zhuoping Yang, Shixin Ji, Heng Huang, Alex K Jones, Jingtong Hu, Yiyu Shi, and Peipei Zhou. Ssr: Spatial sequential hybrid architecture for latency throughput tradeoff in transformer acceleration. In Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pages 55–66, 2024.
[25] Yajing Liu, Ruiqi Chen, Shuyang Li, Jing Yang, Shun Li, and Bruno da Silva. Fpga-based sparse matrix multiplication accelerators: From state-of-the-art to future opportunities. ACM Transactions on Reconfigurable Technology and Systems, 2024.
[26] Seongmin Hong, Seungjae Moon, Junsoo Kim, Sungjae Lee, Minsub Kim, Dongsoo Lee, and Joo-Young Kim. Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation. In 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 616–630. IEEE, 2022.
[27] Shulin Zeng, Jun Liu, Guohao Dai, Xinhao Yang, Tianyu Fu, Hongyi Wang, Wenheng Ma, Hanbo Sun, Shiyao Li, Zixiao Huang, et al. 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, pages 223–234, 2024.
[28] Hongzheng Chen, Jiahao Zhang, Yixiao Du, Shaojie Xiang, Zichao Yue, Niansong Zhang, Yaohui Cai, and Zhiru Zhang. Understanding the potential of fpga-based spatial acceleration for large language model inference. ACM Transactions on Reconfigurable Technology and Systems, 2024.
[29] Jinhao Li, Jiaming Xu, Shan Huang, Yonghua Chen, Wen Li, Jun Liu, Yaoxiu Lian, Jiayi Pan, Li Ding, Hao Zhou, and Guohao Dai. Large language model inference acceleration: A comprehensive hardware perspective, 2024.
[30] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2024.
[31] AMD. Alveo u280 data center accelerator card data sheet. https://docs.amd.com/r/en-US/ds963-u280, 2021.
[32] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.
[33] A Vaswani. Attention is all you need. Advances in Neural Information Processing Systems, 2017.
[34] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. arXiv preprint arXiv:2401.09670, 2024.
[35] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Kimi's kvcache-centric architecture for llm serving. arXiv preprint arXiv:2407.00079, 2024.
[36] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA), pages 118–132. IEEE, 2024.
[37] Aaron Grattafiori et al. The llama 3 herd of models, 2024.
[38] Nvidia. Nvidia h200 tensor core gpu data sheet. https://nvdam.widen.net/s/nb5zzzsjdf/hpc-datasheet-sc23-h200-datasheet-3002446, 2024.
[39] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Proceedings of the 29th Symposium on Operating Systems Principles, page 611–626. Association for Computing Machinery, 2023.
[40] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In Proceedings of the 40th International Conference on Machine Learning, pages 38087–38099. PMLR, 2023.
[41] Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. Owq: Lessons learned from activation outliers for weight quantization in large language models. arXiv preprint arXiv:2306.02272, 2023.
[42] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving. In Proceedings of Machine Learning and Systems, volume 6, pages 196–209, 2024.
[43] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.
[44] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.
[45] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In Advances in Neural Information Processing Systems, pages 27168–27183. Curran Associates, Inc., 2022.
[46] Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. arXiv preprint arXiv:2405.04532, 2024.

[47] Chen Zhang, Shijie Cao, Guohao Dai, Chenbo Geng, Zhuliang Yao, Wencong Xiao, Yunxin Liu, Ming Wu, Lintao Zhang, Guangyu Sun, Zhigang Ji, Runsheng Wang, and Ru Huang. Fine-grained structured sparse computing for fpga-based ai inference. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2024.
[48] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.
[49] Zhaodong Chen, Zheng Qu, Yuying Quan, Liu Liu, Yufei Ding, and Yuan Xie. Dynamic n:m fine-grained structured sparse attention mechanism. In Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, page 369–379. Association for Computing Machinery, 2023.
[50] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019.
[51] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pages 97–110, 2021.
[52] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: transformers for longer sequences. In Proceedings of the 34th International Conference on Neural Information Processing Systems, 2020.
[53] Yubin Qin, Yang Wang, Dazheng Deng, Zhiren Zhao, Xiaolong Yang, Leibo Liu, Shaojun Wei, Yang Hu, and Shouyi Yin. Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction. In Proceedings of the 50th Annual International Symposium on Computer Architecture. Association for Computing Machinery, 2023.
[54] Liqiang Lu, Yicheng Jin, Hangrui Bi, Zizhang Luo, Peng Li, Tao Wang, and Yun Liang. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, page 977–991, 2021.
[55] Bingbing Li, Santosh Pandey, Haowen Fang, Yanjun Lyv, Ji Li, Jieyang Chen, Mimi Xie, Lipeng Wan, Hang Liu, and Caiwen Ding. Ftrans: energy-efficient acceleration of transformers using fpga. In Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design, page 175–180, 2020.
[56] Sheng-Chun Kao, Suvinay Subramanian, Gaurav Agrawal, Amir Yazdanbakhsh, and Tushar Krishna. Flat: An optimized dataflow for mitigating attention bottlenecks. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, page 295–310, 2023.
[57] Tae Jun Ham, Yejin Lee, Seong Hoon Seo, Soosung Kim, Hyunji Choi, Sung Jun Jung, and Jae W. Lee. Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks. In 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), pages 692–705, 2021.
[58] Tae Jun Ham, Sung Jun Jung, Seonghak Kim, Young H. Oh, Yeonhong Park, Yoonho Song, Jung-Hun Park, Sanghee Lee, Kyoung Park, Jae W. Lee, and Deog-Kyoon Jeong. A^3: Accelerating attention mechanisms in neural networks with approximation. In 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), pages 328–341, 2020.
[59] Hongxiang Fan, Thomas Chau, Stylianos I. Venieris, Royson Lee, Alexandros Kouris, Wayne Luk, Nicholas D. Lane, and Mohamed S. Abdelfattah. Adaptable butterfly accelerator for attention-based nns via hardware and algorithm co-design. In 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 599–615, 2022.
[60] Hyungjun Oh, Kihong Kim, Jaemin Kim, Sungkyun Kim, Junyeol Lee, Du-seong Chang, and Jiwon Seo. Exegpt: Constraint-aware resource scheduling for llm inference. In Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, page 369–384, 2024.
[61] Linghao Song, Licheng Guo, Suhail Basalama, Yuze Chi, Robert F. Lucas, and Jason Cong. Callipepla: Stream centric instruction set and mixed precision for accelerating conjugate gradient solver. In Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '23, page 247–258, New York, NY, USA, 2023. Association for Computing Machinery.
[62] Shiyao Li, Yingchun Hu, Xuefei Ning, Xihui Liu, Ke Hong, Xiaotao Jia, Xiuhong Li, Yaqi Yan, Pei Ran, Guohao Dai, et al. Mbq: Modality-balanced quantization for large vision-language models. In Proceedings of the Computer Vision and Pattern Recognition Conference, pages 4167–4177, 2025.
[63] Hongzheng Chen, Niansong Zhang, Shaojie Xiang, Zhichen Zeng, Mengjia Dai, and Zhiru Zhang. Allo: A programming model for composable accelerator design. Proceedings of the ACM on Programming Languages, 8(PLDI):593–620, 2024.
[64] Yuzong Chen and Mohamed S. Abdelfattah. Bramac: Compute-in-bram architectures for multiply-accumulate on fpgas. In 2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pages 52–62, 2023.
[65] AMD. Aurora 64b/66b v13.0 logicore ip product guide, 2024.
[66] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.
[67] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.
[68] HyperAccel. Hyper-accelerated hardware solutions for emerging ai applications. https://hyperaccel.ai/, 2023.
[69] NVIDIA. A tensorrt toolbox for optimized large language model inference. https://github.com/NVIDIA/TensorRT-LLM, 2023.

[70] Shulin, Zeng and Xinhao, Yang and Jun, Liu and Jingtao, Li and Yadong, Dai. Artifact evaluation for fpga 2024 paper #57. https://zenodo.org/records/10462167, 2024.
[71] AMD, HyperAccel. Hyperaccel taps amd accelerator card and fpgas for new ai inference server. https://www.amd.com/content/dam/amd/en/documents/resources/case-studies/hyperaccel-case-study.pdf, 2024.
[72] ShareGPT Teams. Sharegpt. https://sharegpt.com/, 2023.
[73] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. arXiv preprint arXiv:2308.14508, 2023.
[74] Lenovo Press. On-premise vs. cloud: Generative ai total cost of ownership. Technical report, Lenovo Press, 2025.
[75] U.S. Energy Information Administration. Electric power monthly, 2025.
[76] Vu13p fpga price. https://www.win-source.net/products/detail/xilinx-inc/xilinx-inc.-xcvu13p-1fhgb2104e.html, 2025.
[77] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948, 2025.
[78] AMD. Versal ai edge series gen 2. https://www.amd.com/en/products/adaptive-socs-and-fpgas/versal/gen2/ai-edge-series.html, 2025.

Received 30 June 2025; revised 7 September 2025; accepted 22 September 2025