MEADOW: MEMORY-EFFICIENT DATAFLOW AND DATA PACKING FOR LOW POWER EDGE LLMS

Abhishek Moitra [1] **Arkapravo Ghosh** [1] **Shrey Agarwal** [2] **Aporva Amarnath** [3] **Karthik Swaminathan** [3]
Priyadarshini Panda [1]

ABSTRACT

The computational and memory challenges of large language models (LLMs) have sparked several optimization approaches towards their efficient implementation. While prior LLM-targeted quantization, and prior works on sparse acceleration have significantly mitigated the memory and computation bottleneck, they do so assuming high power platforms such as GPUs and server-class FPGAs with large off-chip memory bandwidths and employ a generalized matrix multiplication (GEMM) execution of all the layers in the decoder. In such a GEMM-based execution, data is fetched from an off-chip memory, computed and stored back. However, at reduced off-chip memory capacities, as is the case with low-power edge devices, this implementation strategy significantly increases the attention computation latency owing to the repeated storage and fetch of large intermediate tokens to and from the off-chip memory. Moreover, fetching the weight matrices from a bandwidth constrained memory further aggravates the memory bottleneck problem. To this end, we introduce MEADOW, a framework that significantly reduces the off-chip memory access for LLMs with a novel token-parallel head-sequential (TPHS) dataflow. Additionally, MEADOW applies weight packing, that performs loss-less decomposition of large weight matrices to their unique elements thereby, reducing the enormous weight fetch latency. MEADOW demonstrates 1.5× and 2.5 × lower decode and prefill latency, respectively, compared to a GEMM-based LLM implementation on the low power Xilinx ZCU102 FPGA platform that consumes less than 10W. Additionally, MEADOW achieves an end-to-end latency improvement of over 40%, compared to prior LLM optimization works.

1 INTRODUCTION

The explosive growth of large language models (LLMs) necessitates efficient, low-power hardware solutions to make them accessible across diverse AI applications (Zhang et al., 2024; Minaee et al., 2024; Chang et al., 2024). In particular, there have been several efforts to deploy LLMs across a swath of applications at the edge, ranging from autonomous driving systems (Marcu et al., 2023) to mobile device assistants (Murthy et al., 2024). Even though there have been a few custom ASIC solutions targeting fixed transformer models (Tambe et al., 2023; Park et al., 2024), their significant design/verification complexity and the consequent impact on the time-to-market makes it difficult for them to cater to the rapidly changing nature of the models and their underlying applications. On the other hand, more general-purpose CPU/GPU/TPU solutions deployed on the cloud cannot be replicated on edge devices due to their inherent Size, Weight and Power (SWaP) limitations.

[1]Department of Electrical and Computer Engineering, Yale University, CT, USA [2]IIT Roorkee, Roorkee, India; Work done during internship at Yale [3]IBM Research - Yorktown Heights, Yorktown Heights, NY USA. Correspondence to: Abhishek Moitra <abhishek.moitra@yale.edu.>.

Data-center scale hardware solutions, like the AMD Alveo series (alv), leverage high bandwidth memory (HBM) to handle the intense demands of LLMs, but they also consume over 200 Watts of power. In contrast, platforms like the Xilinx ZCU102 (zcu, a) and Xilinx ZCU104 (zcu, b) offer a reconfigurable, low-power alternative with a sub-10 Watt power budget, making them well-suited for exploring the extensive design space of LLMs, while, balancing power and performance. However, without HBM, these platforms face limitations in available memory bandwidth. This constraint presents a challenge, as the attention computations that drive modern LLMs are highly memory-bound. To mitigate the memory bottleneck in LLMs, techniques like weight quantization (Xiao et al., 2023; Lin et al., 2024; Xu et al., 2024) and sparse computation (Huang et al., 2024; Zhang et al., 2024) have been proposed to reduce data transfer and computational complexity (Wang et al., 2023; Ma et al., 2023). However, these solutions are largely tailored for larger GPUs and/or TPUs. Achieving efficient LLM acceleration on low power-budget devices with restricted memory, calls for a cohesive approach that combines architecture optimization, dataflow restructuring, and parameter compression.

A typical LLM, especially for generative language process-

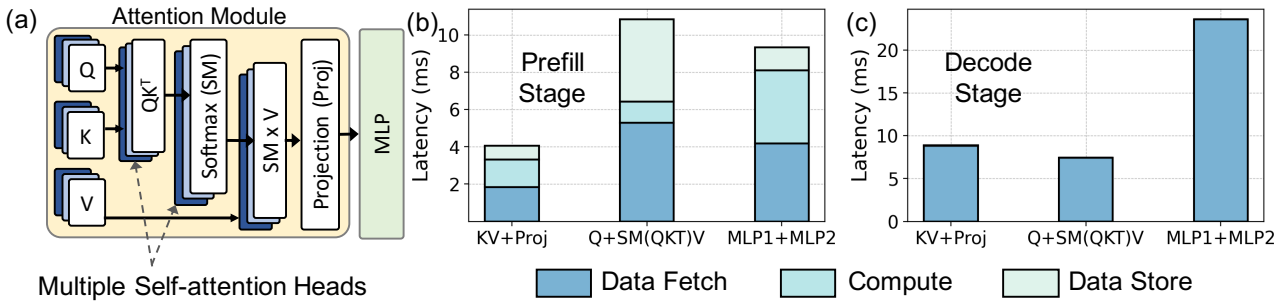


Figure 1. Figure showing the (a) Decoder architecture used in LLMs (b) the prefill latency distribution across data fetch, store and computation across different layers in the decoder (c) the decode latency distributions. During decode, compute and storage latency is negligible compared to the weight and input fetch latency. All latency results are based on OPT-125M LLM implementation on the Xilinx ZCU102 FPGA with off-chip DRAM bandwidth = 12Gbps.

ing, comprises of multiple layers of decoder hierarchy. A decoder architecture (as shown in Fig. 1a) has self-attention and matrix multiplication operations. During inference, the LLM operates in two stages: *Prefill* and *Decode*. In the prefill stage, a user-provided prompt is decomposed into multiple tokens. These tokens simultaneously undergo matrix multiplications with multi-dimensional weights to yield Q, K and V outputs. Subsequently, the Q, K and V values undergo fine-grained spatial correlations by means of SM (QK^T)xV operations in multiple self-attention heads, where SM denotes a Softmax operation. The attention outputs are finally projected to higher dimension space by the projection (Proj) and MLP layers. Post prefill, the LLM enters the decode stage, where it predicts subsequent output tokens one-by-one.

In most prior works, the SM (QK^T) xV layers are executed in the form of a generalized matrix multiplication (GEMM) operation (Zeng et al., 2024; Wang et al., 2023; Huang et al., 2024). Here, the input matrices for each self-attention head are fetched from the off-chip memory, processed in the GEMM array, and the output is stored back to the off-chip memory. As shown in Fig. 1b, under limited off-chip memory bandwidth (12 Gbps), this repeated data transfer significantly increases the latency in the prefill stage, where larger input sizes exacerbate memory access demands. During the decode stage, however, the input size is much smaller, reducing compute and data storage overheads to a negligible fraction, while the weight fetches dominate latency (Fig. 1c). Thus, optimizing on memory accesses through efficient compute dataflow in both the prefill and decode stages is essential to reduce the overall latency.

To address the above challenges, we introduce the MEADOW framework. During the prefill and decode stage, MEADOW executes the KV, Proj and MLP layers in the GEMM mode while, the Q, QK^T, SM, and SMxV layers are executed with a novel Token-Parallel Head-Sequential (TPHS) dataflow which performs effective layer pipelining and significantly reduces the off-chip data fetches and storage latency. To further mitigate the latency and bandwidth overhead of weight fetches, MEADOW implements Weight Packing, which compacts the weight matrix by transferring only its unique elements, significantly minimizing weight transfer volume. Additionally, MEADOW applies bit-packing techniques on the weights to maximize memory bandwidth utilization, enhancing memory efficiency.

The key contributions of our work are as follows:

1. We propose MEADOW that uses a novel Token Parallel Head Sequential (TPHS) dataflow to compute the SM (QK^T)xV layers in pipeline, significantly reducing the volume of data transfers to and from off-chip memory.

2. We introduce Weight Packing, a technique that decomposes LLM weight matrices into unique elements to minimize weight fetch latency at prefill and decode stages. Additionally, to further accelerate weight fetches and maximize DRAM bandwidth efficiency, we implement bit-packing to compactly store and transfer weight data. Weight packing is an approximation-less technique that yields loss-less accuracy performance.

3. We evaluate MEADOW on the ZCU102 FPGA with a peak power budget of 10W across varying off-chip DRAM bandwidths and input token lengths on state-of-the-art OPT-125M and OPT-1.1B LLM models. MEADOW achieves 2.5× and 1.5× lower prefill and decode latency compared to GEMM-based implementations for 1-6 Gb/s data bandwidth ranges. MEADOW also achieves over 40% end-to-end latency improvement compared to prior LLM optimization works.

4. We demonstrate the generalizability of MEADOW across vision transformer (ViT) benchmarks, achieving 1.6× lower inference latency compared to GEMM-based ViT implementations. We also demonstrate how

MEADOW can be applied to multiple FPGA configurations with varying PE sizes and memory bandwidth.

2 Related Work

Data compression techniques: Weight and input quantization is a widely adopted approach for data compression in LLMs. Works such as SmoothQuant (Xiao et al., 2023), AWQ (Lin et al., 2024), and LlamaF (Xu et al., 2024) apply fake quantization methods to lower off-chip data transfers, and dequantize the compressed inputs and weights during computation to maintain good accuracy. A recent work MECLA (Qin et al., 2024) applies a sub-matrix partitioning technique wherein, different sub-matrices within a larger matrix is approximated as a function of a base sub-matrix.

Sparse Computations: Sparse computation techniques leverage the inherent dynamic sparsity of LLMs to reduce computation. Unstructured sparsity, as implemented in methods like ELSA (Huang et al., 2024; Fang et al., 2022; Chen et al., 2023) with N:M sparsity, selectively prunes non-essential connections, effectively reducing computational load. FlightLLM (Zeng et al., 2024) implements N:M sparse computation using FPGA-based accelerators with HBM to address memory bottlenecks.

Structured pruning addresses the limitations of unstructured pruning by removing entire blocks or groups of computations. For example, token compression in CTA (Wang et al., 2023) reduces memory and compute demands by compressing less critical tokens. Gradient-based pruning, as used in LLM Pruner (Ma et al., 2023), selectively prunes attention heads based on gradient information, focusing computational resources on essential parts of the model. ALISA (Zhao et al., 2024) focuses on retaining tokens that are crucial towards generating new tokens via a sparse window attention technique. FACT (Qin et al., 2023) focuses on performing eager computation of attention tokens at minimal computation overhead and performing sparse computations for subsequent layers.

Parallel research directions towards designing more hardware efficient transformer architectures are also being developed. EdgeBERT (Tambe et al., 2021), PIVOT (Moitra et al., 2024b) and TReX (Moitra et al., 2024a) use entropy of inputs to perform dynamic voltage scaling, attention skipping and reuse, respectively to achieve hardware efficiency. FlexLLM (Miao et al., 2024) introduce a unique inference and parameter-efficient finetuning to achieve efficient yet, highly accurate LLMs.

MEADOW is an orthogonal solution to prior techniques, introducing architectural and dataflow innovations along with weight packing to optimize weight fetch latency. By restructuring the dataflow and enhancing memory access patterns, MEADOW minimizes latency in retrieving weights, addressing memory bottlenecks in low memory bandwidth hardware without sacrificing model accuracy.

3 MEADOW Architecture

MEADOW follows a tiled architecture as shown in Fig. 2a containing multiple processing elements (PEs), modules for layer normalization (LN), softmax operations (SM) and non-linear (NL) activation functions, such as, ReLU/GeLU. Each PE contains several multipliers and accumulators to carry out the multiply-accumulate operations. For computation, the input data is fetched from the off-chip DRAM to the Input block RAM (BRAM). The raw input values are directly transferred to the respective input register files (RF) of the PEs. Since, MEADOW applies an additional weight packing to reduce off-chip weight fetches, the Weight BRAM stores the packed and encoded weight values which first needs to be processed by the Weight Unpacking and Index Look-up (WILU) Module. The WILU module reads data from the Weight BRAM and sends the data to the respective weight RFs of the PEs. The outputs from each PE are stored back to the output BRAM. All communications between BRAM and PE, SM, LN and NL modules are enabled by the network on chip (NoC) interconnect. The NoC additionally handles data communication between PEs and SM modules to facilitate the TPHS dataflow, defined in Section 4.

Hybrid PE for GEMM and Pipelined Execution: MEADOW employs a dual execution strategy: GEMM mode for the `KV`, `Proj`, and `MLP` layers, and the TPHS dataflow for the `Q`, $\texttt{QK}^{\texttt{T}}$, `SM`, and `SMxV` layers, enabling pipelined execution that minimizes data fetch and store latency. To support both GEMM and pipelined modes seamlessly, MEADOW utilizes a hybrid PE architecture, designed for flexible execution across modes. The PE shown in Fig. 2b, integrates a multiply-accumulate (MAC) unit, input, weight, and output register files (RF), along with a pipeline register (PREG). All RFs and the pipeline registers are double-buffered to minimize data fetch and store latency (Moitra et al., 2024b).

For the GEMM mode, data from the input and weight BRAMs are loaded into the input and weight RF, respectively. These data values are fetched and processed in the MAC array and the outputs are stored in the output RF. Once the output RF reaches capacity, the data is transferred to the output BRAM via the NoC. In contrast, for the pipelined mode, weights are loaded from the BRAM into the weight RF while the inputs are fetched directly from the pipeline register. The Input BRAM remains inactive during the pipelined mode of operation. After the MAC operation, the outputs are transferred directly through the NoC to the pipeline register of a target module (such as the softmax unit or another PE) in the subsequent pipeline stage.

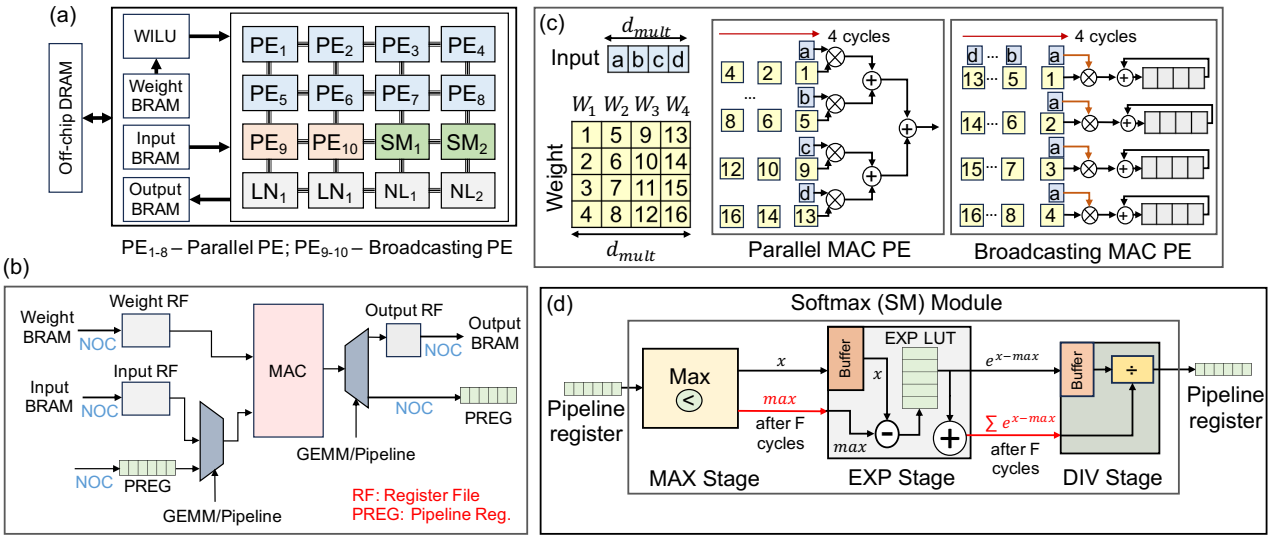


Figure 2. (a) Tiled architecture of MEADOW containing parallel and broadcasting processing elements (PEs), pipelined softmax (SM) module, modules for layer normalization (LN) and non-linear activation functions like ReLU/GeLU (NL). (b) The hybrid PE architecture capable of operating in GEMM and pipelined modes. (c) Architecture and execution flow of a parallel and broadcasting MAC PE. (d) The pipelined softmax (SM) module.

Parallel and Broadcasting PE: MEADOW's tiled architecture contains a mix of Parallel MAC and Broadcasting MAC PEs (for example $\texttt{PE}_{1-8}$ = Parallel and $\texttt{PE}_{9-10}$ are Broadcasting MAC PEs as shown in Fig. 2a). As shown in Fig. 2c, both parallel and broadcasting MAC PEs use an array of multipliers but use different accumulation strategies. The Parallel MAC PE incorporates an adder tree, allowing it to multiply all elements along the multiplication dimension ($\texttt{d}_{\texttt{mult}}$) in a single cycle. In contrast, the Broadcasting MAC PE features accumulators (registers coupled with adders), enabling it to broadcast each input element along $\texttt{d}_{\texttt{mult}}$ across all corresponding output channels and perform multiplication and accumulation sequentially over $\texttt{d}_{\texttt{mult}}$ cycles. The Parallel MAC and Broadcasting MAC PEs are essential for facilitating the TPHS dataflow, described in Section 4.

Pipelined Softmax Module (SM Module): The numerically stable softmax computation of a given token is shown in Equation 1.

$$SM = \frac{e^{x_i - max}}{\Sigma_i e^{x_i - max}} \tag{1}$$

The computation requires three sequential stages: 1) finding the maximum across all the features in the token, 2) computing the exponent and the summation of all exponents and, 3) f inally, dividing each exponent value with the exponent summation. Due to the sequential nature of the softmax stages, it is latency intensive. To this end, MEADOW pipelines the three stages across tokens to improve the softmax computation throughput. As shown in Fig. 2d, the SM Module consists of three pipelined stages `MAX`, `EXP` and `DIV`. Each stage processes a token feature-by-feature over `F` cycles, where `F` is the number of features in the token. The `MAX` stage compares the feature values and returns the maximum value at the end of `F` cycles. Subsequently, the values are written to the `EXP` stage buffer. In the `EXP` stage, the maximum value output from the `MAX` stage is subtracted from each feature and the exponent values are computed. For hardware efficiency, the exponent is computed using the `EXP LUT` lookup table. Simultaneously, the exponent values are summed up and are stored in the DIV stage buffer. Finally, in the `DIV` stage the exponent values are fetched from the `DIV` stage buffer and divided by the exponent summation value.

4 TPHS Dataflow

To overcome the memory bound implementation of `Q`, $\texttt{QK}^{\texttt{T}}$, `SM(QKT)` and `SMxV` operation, MEADOW uses the token parallel head sequential (TPHS) dataflow. The TPHS dataflow shown in Fig. 3a, pipelines all the computations for each attention head in parallel across multiple tokens. In the example in Fig. 3a, we show how attention head 1 (`H1`) is computed for input tokens $\texttt{IP}_1$ and $\texttt{IP}_2$. The TPHS dataflow requires the following data from the off-chip DRAM to be stored before computation- the input tokens $\texttt{IP}_1$ and $\texttt{IP}_2$ of size `1xD` each, the $\texttt{K}_{\texttt{H1}}$, $\texttt{V}_{\texttt{H1}}$ precomputed values for head H1 of size `TxHD`, where `T` and `HD` are the total number of input tokens and head dimension, respectively. Additionally, for the $\texttt{Q}_{\texttt{H1}}$ computation, the $\texttt{W}_{\texttt{Q, H1}}$ matrix of dimensions `DxHD` are required.

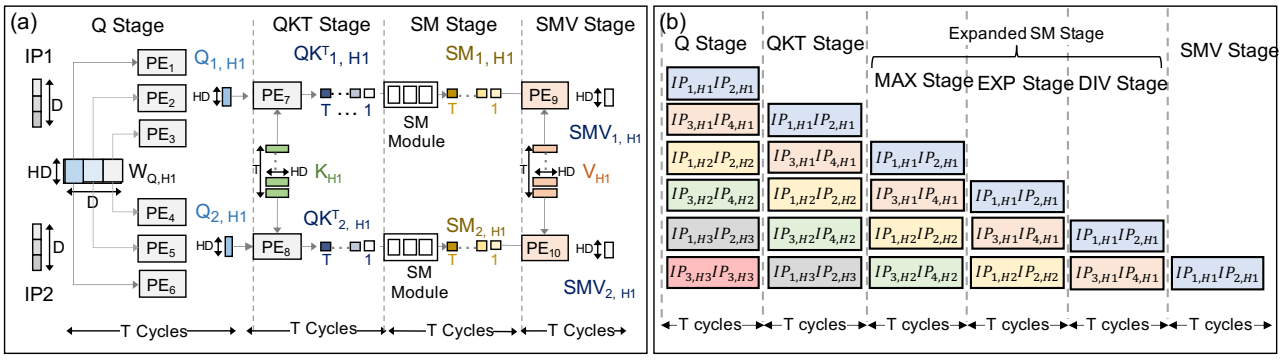
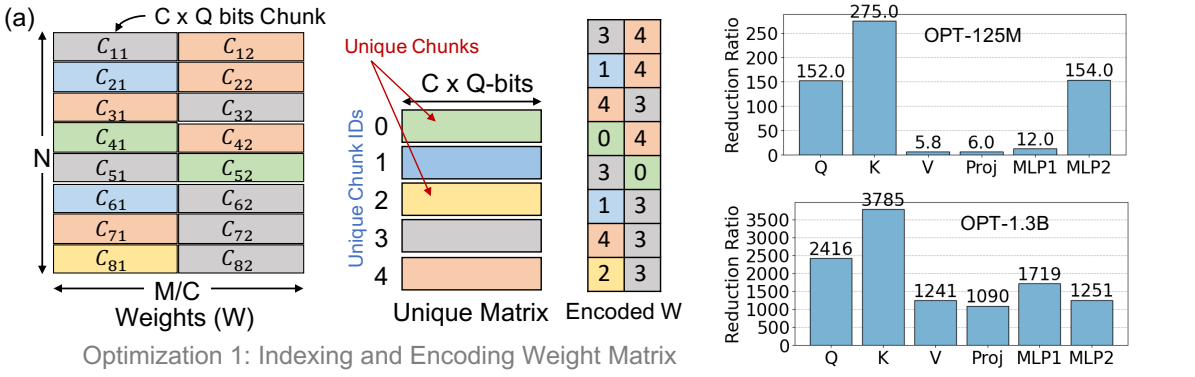
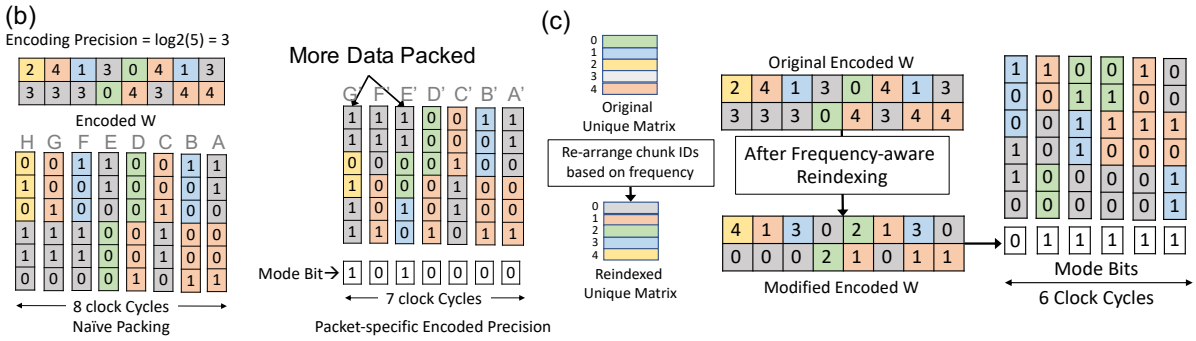


Figure 3. Figure showing an example of (a) token parallel head sequential (TPHS) dataflow with two input tokens being processed parallely (b) The pipelined execution of a transformer with 3 heads (H1–H3) and 4 input tokens (IP_{1-4}).



Optimization 1: Indexing and Encoding Weight Matrix



Optimization 2: Packet-specific Encoding Precision Optimization 3: Frequency-aware Reindexing

Figure 4. Figure showing (a) process of generating the unique matrix and the trends in the reduction ratios for OPT-125M and OPT-1.3B LLM models across different layers in the decoder. Reduction ratios are averaged across all the decoder layers. (b) packet-specific encoding precision and (c) frequency-aware reindexing to further optimize the DRAM bandwidth.

IP_1 and IP_2 are multiplied by $W_{Q,H1}$ parallelly in PE_{1-3} and PE_{4-6}, respectively. This results in $Q_{1,H1}$ and $Q_{2,H1}$ for the two input tokens. $Q_{1,H1}$ and $Q_{2,H1}$ data is sent to the pipeline registers of PE_7 and PE_8, respectively where they are multiplied with T tokens of K_{H1} resulting in $QK^T_{1,H1}$ and $QK^T_{2,H1}$ values over T cycles. At each cycle, the QK^T outputs are sent to the MAX stage of softmax module which returns the maximum across all the QK^T values at the end of T cycles. Subsequently, these values are forwarded to the EXP stage and the DIV stage which finally yield the SM values over T cycles. In Fig. 3a, the MAX, EXP and DIV stages are combined into SM stage for simple visualization. The respective softmax outputs are sent to the pipeline registers of the broadcasting PEs PE_9 and PE_{10} to compute the SMxV output. Here, the SM outputs are multiplied with V_{H1} tokens over T cycles to yield $SMV_{1,H1}$ and $SMV_{2,H1}$ outputs for both tokens. The $SMV_{1,H1}$ and $SMV_{2,H1}$ outputs are stored to the off-chip DRAM. As shown in Fig. 3a, each stage requires T clock cycles.

Fig. 3b shows an example of the pipelined execution of TPHS dataflow. Here, we consider a transformer having 3 self-attention heads with 4 tokens and two tokens being simulatenously processed. Additionally, we show the expanded stages inside the SM module for better visualization. In the TPHS dataflow, first all H1 self-attention heads are computed for every input token before proceeding to the computation of H2. This minimizes the amount of back-and-forth data transfers of the W_Q, K and V matrices thereby minimizing additional latency overhead.

5 Weight Packing

5.1 Creating the Unique Matrix

Let W be a matrix of trained weight values with dimensions NxM, where M represents the inner product dimension. As shown in Fig. 4a, the inner dimension M is divided into chunks of size C, where each element in C is a Q-bit value based on the quantization of the weight matrix. Next, as illustrated in Fig. 4a, a `Unique Matrix` is generated, containing the unique chunks, each assigned a unique ID. These chunk IDs are used to encode the weight matrix, resulting in the creation of the `Encoded W` matrix. To intuitively understand the amount of redundancy in the LLM weight matrices, we define the reduction ratio as the ratio between the total number of chunks in the encoded W matrix ($N \times M/C$) and the number of unique chunks. Higher reduction ratio signifies more redundancy and vice-versa. As seen in Fig. 4a, for the decoder weights of OPT-125M and OPT-1.3B the reduction ratio varies in the order of 10^2 to 10^3 suggesting high redundancy in the weight matrices.

5.2 Packet-specific Encoding Precision

To improve the DRAM bandwidth efficiency, multiple elements of the encoded W matrix are grouped together to form a packet and transferred from the DRAM for processing. As shown in Fig. 4b with naive data packing, all the packets use the same data precision to represent the encoded weights. The precision here is determined by the maximum number of unique chunks in the unique matrix (5 as in the Fig. 4a). However, using homogeneous bit-precision across packets lead to inefficiencies, as cycles are wasted transmitting low-precision encoded values that could otherwise be represented with fewer bits. For example, packets E and F use 3-bit precision to represent 2-bit numbers.

To this end, we employ packet-specific bit-precision to represent the encoded values, where each packet is assigned an optimal precision to maximize packing efficiency. As depicted in Fig. 4b, employing packet-specific bit-precision allows low-bit encoded values to be packed together more effectively, thereby reducing the number of cycles required for transmission. The encoding precision for each packet is determined by the maximum encoded value in the respective packet. Additional mode bits are now used to determine the bit-precision of each packet (for example 3-bits for packet A′ and 2-bits for packets E′, G′). Packets with mode = 0 and mode = 1 use 3-bits and 2-bits to represent the encoded values, respectively. The mode bits will be used by the WILU module to unpack the grouped encoded values. Packet-specific encoding allows packing more data per packet thereby improving the DRAM bandwidth efficiency.

5.3 Frequency-aware Re-indexing

As illustrated in Fig. 4c, frequently occurring chunk IDs in the encoded W matrix (e.g., chunk ID = 3) may necessitate higher precision, which can limit the efficiency of bit packing. In frequency-aware re-indexing, the chunk IDs are re-assigned to each unique chunk based on their frequency of occurrence *i.e.,* chunk IDs appearing more frequently are assigned lower chunk ID. For instance, in the example presented in Fig. 4c, chunk IDs [0, 1, 2, 3, 4] with frequencies [2, 2, 1, 6, 5] are re-assigned new chunk IDs [2, 3, 4, 0, 1]. This approach increases the proportion of low-precision chunk IDs in the encoded W matrix, resulting in efficient bit packing and thereby reducing transfer cycles. The modified encoded W and the reindexed unique matrix are transferred from the DRAM for processing.

5.4 Weight unpacking and Index Look-up Module

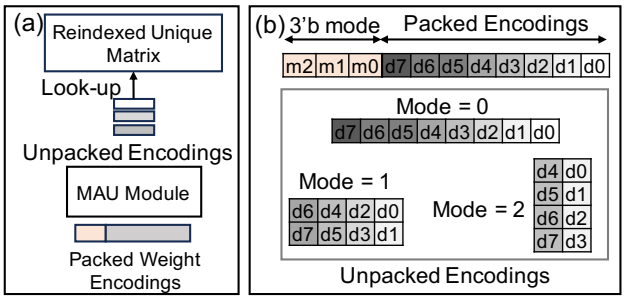


Figure 5. (a) The WILU Module (b) The mode-aware unpacking (MAU) module.

Fig. 5a shows the execution of the WILU module. The WILU module reads the encoded and packed weight values from the weight BRAM as discussed in Section 3. A packet read from the weight BRAM contains mode bits and packed encoded weight values. The mode aware unpacking (MAU) module unpacks the packed encodings based on the mode as shown in Fig. 5b. For example, for an 8-bit packed encoding, d0 to d7 is unpacked in 1, 2 and 4-bit values for modes 0, 1 and 2, respectively. The unpacked encodings are used to look up the reindexed unique matrix to get the actual weight values that are sent to the weight RF of the respective PE through the NoC.

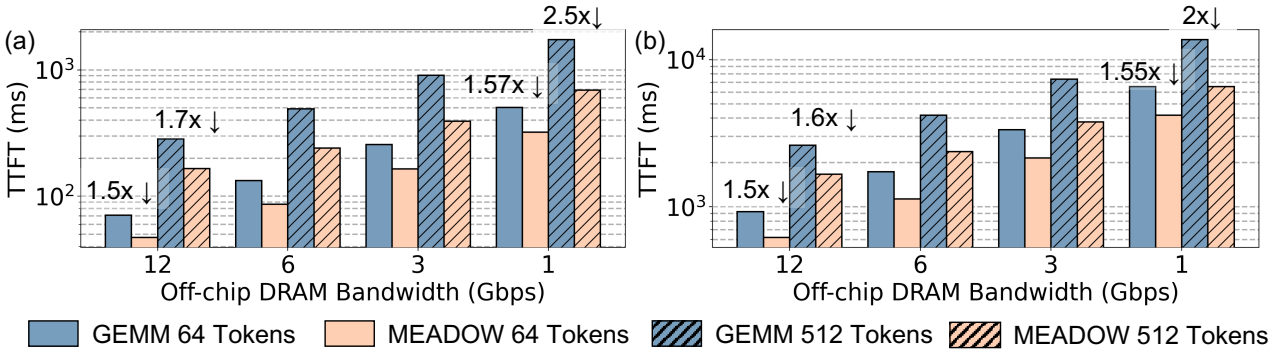


Figure 6. Time to first token (TTFT) Comparison of MEADOW with GEMM-based decoder implementation of the (a) OPT-125M and (b) OPT-1.3B LLM models on the ZCU102 FPGA with varying off-chip DRAM bandwidths. The evaluations are performed with 64 and 512 tokens during the prefill stage.

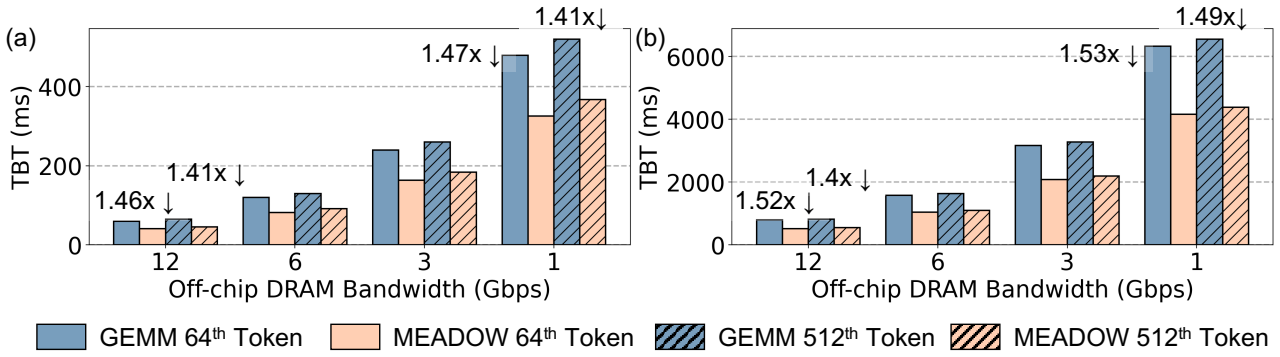


Figure 7. Time between tokens (TBT) comparison of MEADOW with GEMM-based decoder implementations of the (a) OPT-125M and (b) OPT-1.3B LLM models on the ZCU102 FPGA with varying off-chip DRAM bandwidths. For all cases the number of prefill tokens set to 512. The TBT is then measured for the 64th and 512th predicted token in the decode stage.

6 Results and Analyses

6.1 Experiment Setup

LLM Models and Datasets: For benchmarking MEADOW, we use the OPT-125M and OPT-1.3B LLM models (Zhang et al., 2022) finetuned on the LAMBADA dataset using zero-shot adaptation with Smoothquant (Xiao et al., 2023) post-training quantization. The weights and inputs are quantized to 8-bit precision. The 8-bit weight and input quantized OPT-125M and OPT-1.1B models achieve **60.7% and 69.7% accuracy** on the LAMBADA dataset.

Xilinx ZCU102 FPGA Implementation: For hardware evaluation, we implement the hybrid GEMM-Pipelined architecture of MEADOW on the Xilinx ZCU102 FPGA using the hardware parameters shown in Table 1. The implementation uses 150K LUT, 845 BRAM and 2034 DSP resources. To maximize the number of PEs, we utilize both LUTs and the DSP blocks. Additionally, register files and pipeline registers are implemented using the LUT-based registers.

GEMM Baseline: To benchmark prefill and decode latency, we use the GEMM baseline. The GEMM baseline is realized by operating the MEADOW architecture in fully GEMM mode. Here, all the layers in the decoder `Q`, `K`, `V`, $\texttt{QK}^{\texttt{T}}$, `SMxV`, `Proj` and `MLP` are executed in the GEMM mode. This captures the standard execution pattern that is followed in all prior LLM optimization works.

Parameter	Value
#Parallel & #Broadcasting PEs	84, 12
#Multipliers per PE	64
#SM, #LN & #ReLU Modules	84, 8, 8
Weight, Input & Output BRAM Size	1MB, 1MB & 1MB
Weight, Input & Output RF Size	4KB, 4KB & 4KB
Clock Frequency	100 MHz

Table 1. Hardware Parameter Table for ZCU102 FPGA Evaluation

Prefill and Decode Latency Measurement: We use time to first token (TTFT) and time between tokens (TBT) to measure the prefill and decode latency, respectively. TTFT measures the time from when a prompt is submitted to the LLM until the first generated token is produced. It reflects the initial processing delay to infer the context of a given prompt by the LLM. TBT measures the latency of

generating the N^{th} token after the LLM has produced $N-1$ tokens post the prefill stage (Zhang et al., 2024).

MEADOW Operation Modes: During the prefill and decode stage, we execute the TPHS dataflow for the Q+SM(QK^T)xV layers and GEMM is used for the remaining K, V, Proj and MLP layers. Weight Packing is applied in both stages. Note, during Decode, there is a marginal latency speedup with TPHS compared to GEMM operation for Q+SM(QK^T)xV since the input token size is 1. As we will see later, the decode stage latency gains are primarily stemming from weight packing.

6.2 Prefill and Decode Latency Improvements

Prefill: Fig. 6a and Fig. 6b compares the TTFT achieved by MEADOW and GEMM-based OPT-125M and OPT-1.3B LLM models for varying DRAM bandwidths. At DRAM bandwidth of 12 Gbps, MEADOW achieves 1.5×-1.7× and 1.5-1.6× for OPT-125M and OPT-1.3B LLMs, respectively across different number of prefill tokens. At a low DRAM bandwidth of 1 Gbps, MEADOW achieves 1.57-2.5× and 1.55-2× lower TTFT compared to GEMM implementations for OPT-125M and OPT-1.3B LLM models, respectively.

Decode: Fig. 7a and Fig. 7b compare the TBT achieved by MEADOW and GEMM-based approaches on the OPT-125M and OPT-1.3B LLM models, across varying DRAM bandwidths. For predicting the 64th and 512th token at 12 Gbps DRAM bandwidth, MEADOW reduces TBT by 1.4-1.46× and 1.4-1.52× for the OPT-125M and OPT-1.3B models, respectively. When operating at a constrained DRAM bandwidth of 1 Gbps, MEADOW achieves a 1.4×-1.47× reduction in TBT for the OPT-125M model and a 1.5×-1.53× reduction for the OPT-1.3B model.

The latency reduction observed in MEADOW for both prefill and decode stages stems from the targeted optimizations in data fetch and storage cycles. In GEMM-based implementations, executing the Q+SM(QK^T)xV layers during the prefill stage requires fetching weights and intermediate values from off-chip DRAM, performing matrix multiplications, and storing outputs back to DRAM. These data transfers impose substantial latency, especially as the size of the intermediate outputs scales directly with the number of attention heads and prefill stage tokens. This latency is exacerbated when the DRAM bandwidth is constrained (as illustrated in Fig. 8a and Fig. 8b).MEADOW's TPHS dataflow with pipelined operations within the Q+SM(QK^T)xV layers minimizes the number of off-chip memory accesses yielding a significant reduction in latency. For the KV+Proj and MLP layers, where data fetches are dominated by weight matrix transfers, the introduction of weight packing further reduces the latency by decreasing the volume of weight data fetched from the off-chip DRAM.

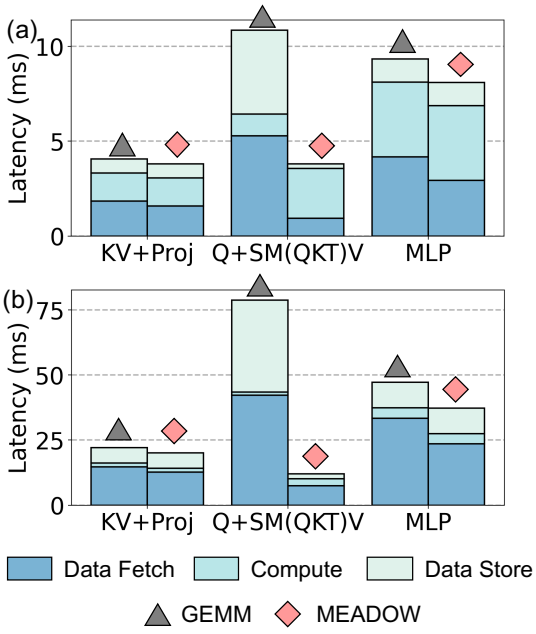


Figure 8. Prefill latency distribution for data fetch, compute and storage with 512 tokens at (a) 12 Gbps and (b) 1 Gbps off-chip DRAM bandwidth. The latency distribution is shown for one decoder layer of the OPT-125M LLM.

During the decode stage, only a single token is processed at a time, significantly reducing input fetch and output storage demands compared to the prefill stage with its large pool of tokens. This limited data transfer, shown in Fig. 9a and Fig. 9b, makes weight data fetching the primary bottleneck. MEADOW is able to reach lower decode latency due to the weight packing strategy that reduces weight fetch latency.

6.3 Efficacy of the Weight Packing Strategy

Indexing reduces a large weight matrix to unique chunk values and represents the weight matrix in terms of the unique chunk IDs. Fig. 10a, analyses the latency improvements over different packing optimizations for the first MLP layer weights of decoder 1 of the OPT-125M LLM. The MLP1 weight is decomposed into 1272 unique chunks leading to a 11-bit encoded W precision. These 11-bit encoded W values are now grouped together into packets to improve the DRAM bandwidth efficiency.

As seen in Fig. 10a, with naive packing, a latency improvement of merely 1.4× is achieved as several low-bit precision encoded W values are represented with 11-bit values. Upon using packet specific encoding precision, a 1.54× lower latency is observed as multiple low bit encoded W values are grouped per packet which reduces the number of data fetch cycles.

The limited improvements in memory fetch latency with naive and packet-specific grouping arises due to the frequent

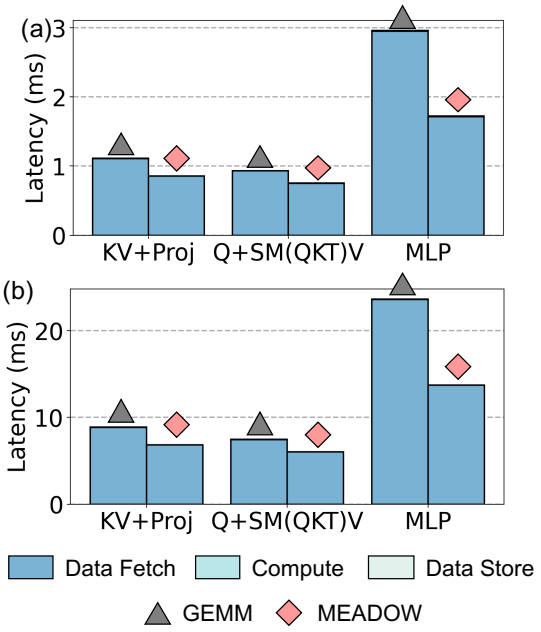


Figure 9. Decode latency distribution for data fetch, compute and storage at (a) 12 Gbps and (b) 1 Gbps off-chip DRAM bandwidth for one decoder layer of the OPT-125M LLM. The latency is shown for predicting the 64th token with 512 tokens at the prefill stage. The compute and store latencies are negligibly small compared to data fetch latency.

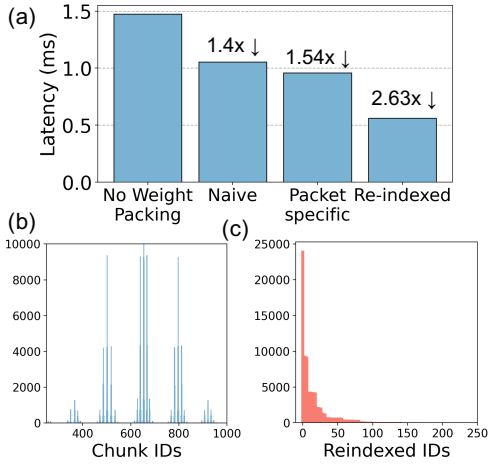


Figure 10. (a) Latency comparison of weight matrix transfer for 3 different weight packing optimizations. 1) indexing + naive data packing (Naive), 2) Indexing + packet specific encoding precision (Packet specific) and 3) frequency aware re-indexing + packet-specific encoding precision. (b) Histogram of the unique chunk IDs (shown for Chunk IDs between 200 and 1000) (c) histogram of the chunk IDs after performing frequency-aware re-indexing.

occurrence of high-value chunk IDs, which hinders effective grouping of the encoded W values, as illustrated in Fig. 10b. To this end, frequency-aware reindexing increases the number of low bit chunk IDs (Fig. 10c) and thereby improves the packing efficiency leading to 2.63× lower weight fetch latency.

6.4 Comparison with Prior Works

	CTA (Wang et al., 2023)	FlightLLM (Zeng et al., 2024)	**MEADOW (Ours)**
KV, Proj, MLP	GEMM	GEMM	**GEMM**
Q, SM(QKT)V	GEMM	GEMM	**TPHS**
Quantization	W8A8	W8A8	**W8A8**
Weight Packing	✗	✗	✔

Table 2. Evaluation settings for prior work comparison.

We implement prior state-of-the-art LLM optimization approaches- CTA (Wang et al., 2023) and FlightLLM (Zeng et al., 2024) on the MEADOW architecture with implementation parameters shown in Table 1. As seen in Table 2, CTA (Wang et al., 2023) and FlightLLM (Zeng et al., 2024) execute all layers in the decoder in the GEMM mode. For fairness, the activations and weights in all works are maintained at 8-bit precision. MEADOW is implemented with weight packing and the TPHS dataflow for the $\texttt{Q+SM}(\texttt{QK}^{\texttt{T}})\texttt{xV}$ layers for both prefill and decode stages. The $\texttt{K}$, $\texttt{V}$, $\texttt{Proj}$ and $\texttt{MLP}$ layers are executed in the GEMM mode. Fig.

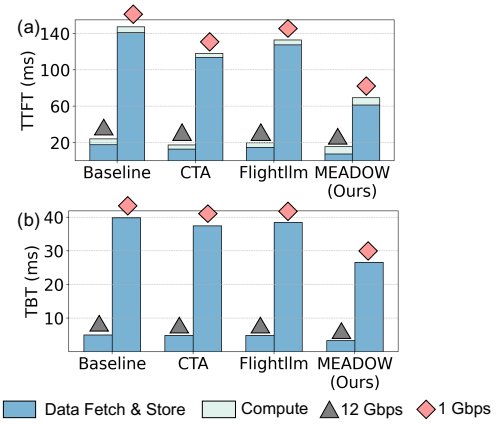


Figure 11. Figure comparing the (a) TTFT and (b) TBT latency of prior state-of-the-art LLM optimization works with MEADOW at different off-chip DRAM bandwidths.

11a and Fig. 11b compares the TTFT and TBT latency of prior works with MEADOW. CTA (Wang et al., 2023) employs token compression to mitigate data redundancy, aiming to reduce memory and computational load by processing essential tokens only. While this approach decreases compute cycles, output storage, and input fetch latency in the $\texttt{Q+SM}(\texttt{QK}^{\texttt{T}})\texttt{xV}$ layers, the intermediate values for the remaining significant tokens still require fetching and storage in off-chip DRAM. Under constrained memory bandwidth, the latency involved in weight and token fetch/storage creates a substantial bottleneck, which limits CTA's overall benefits during both prefill and decode stages.

FlightLLM (Zeng et al., 2024), on the other hand, leverages unstructured N:M sparse acceleration architecture to

cut down computations. While unstructured sparsity can lower compute requirements, it leaves input fetch latency largely unoptimized, and like CTA, FlightLLM does not apply any weight packing technique. To mitigate intermediate storage requirements during $\mathrm{Q{+}SM}(\mathrm{QK^T})\times\mathrm{V}$ operations, FlightLLM utilizes on-chip storage at decode time. However, since output storage latency during decode is negligible, as illustrated in Fig. 11b, weights remain the dominant bottleneck, restricting overall performance gains.

Evidently, prior methods perform prefill and decode with unoptimized weight matrix sizes and only partially eliminate intermediate data fetch and storage cycles during the $\mathrm{Q{+}SM}(\mathrm{QK^T})\times\mathrm{V}$ operations. This partial approach limits their effectiveness, particularly under low-memory bandwidth constraints, where repeated fetches of intermediate values and weights cause latency bottlenecks. MEADOW offers architectural support and the TPHS dataflow innovation to completely eliminate the data fetch and storage latency of the $\mathrm{Q{+}SM}(\mathrm{QK^T})\times\mathrm{V}$ layers. Additionally, weight packing further reduces the latency of fetching the weight matrix. This translates to a 40% reduction in the end-to-end latency with MEADOW compared to FlightLLM and CTA on ZCU102 FPGA-based OPT-125M implementation.

6.5 Choosing between GEMM & TPHS Dataflow

From Fig. 12a, it is observed that the choice of GEMM and TPHS dataflow for the $\mathrm{Q{+}SM}(\mathrm{QK^T})\times\mathrm{V}$ layers is dependent on the number of PEs and the off-chip DRAM bandwidth. For high memory bandwidth scenarios, (BW:51, PE:14) and (BW:51, PE:96) GEMM is the dataflow choice. In contrast, TPHS is suitable for low memory bandwidth configurations (Fig. 12b). This study justifies our framework as a suitable choice for deployment on a range of low memory capability edge devices.

6.6 ViT Latency Improvements with MEADOW

We also show the generality of MEADOW for ViT models. Vision transformers (ViTs) process multiple tokens together like the prefill stage of an LLM. With combined TPHS/GEMM dataflow and weight packing, MEADOW achieves 1.5-1.6× lower inference latency on the DeiT-S and DeiT-B (Touvron et al., 2021) models trained on the ImageNet dataset (Deng et al., 2009) across different off-chip DRAM bandwidths.

7 Conclusion

This work proposes MEADOW- targeting the latency intensive data fetch/store cycles of intermediate outputs and weights through the TPHS dataflow and Weight Packing to achieve 1.5× and 2.5× lower decode and prefill latency compared to GEMM-based implementations. MEADOW is crafted to achieve low latency LLM execution at highly constrained off-chip DRAM bandwidths achieving over 40% end-to-end latency improvement compared to prior LLM optimization works. Additionally, we demonstrate the versatility of MEADOW by applying it towards ViT implementations. This typically makes MEADOW suitable for low power edge applications such as autonomous driving and mobile chatbots for both vision and NLP tasks.

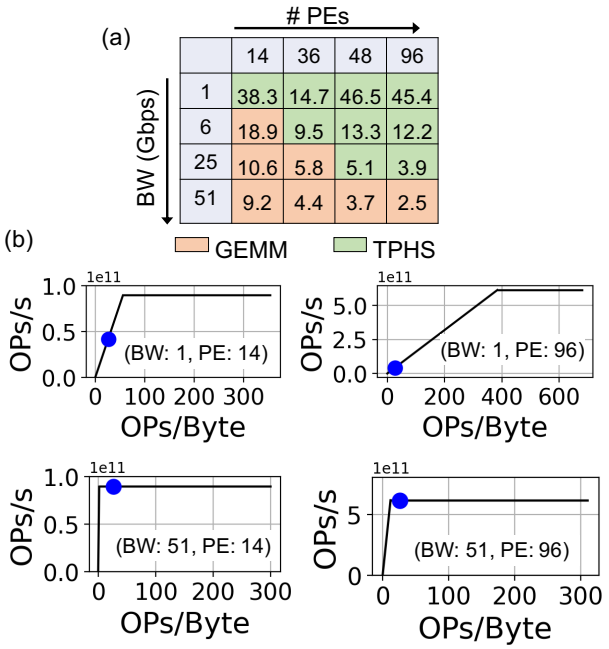


Figure 12. (a) Table showing optimal dataflow chosen for executing the $\mathrm{Q{+}SM}(\mathrm{QK^T})\times\mathrm{V}$ layers and the corresponding optimal prefill latency obtained for the OPT-125M LLM model. (b) The roofline plots for different (Bandwidth (BW), PE) configurations (1,14), (1,96), (51,14) and (51,96).

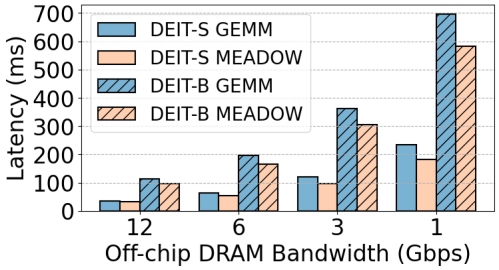


Figure 13. DeiT-S and DeiT-B ViT inference latency improvements with MEADOW compared to GEMM-based implementations on the ZCU102 FPGA.

8 Acknowledgment

This work was supported in part by CoCoSys, a JUMP2.0 center sponsored by DARPA and SRC, the National Science Foundation (CAREER Award, Grant #2312366, Grant #2318152), the DARPA Young Faculty Award and the DoE MMICC center SEA-CROGS (Award #DE-SC0023198).

REFERENCES

AMD Alveo™ Adaptable Accelerator Cards. URL https://www.amd.com/en/products/accelerators/alveo.html.

Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit, a. URL https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html.

Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit, b. URL https://www.xilinx.com/products/boards-and-kits/zcu104.html.

Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., et al. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45, 2024.

Chen, Z., Qu, Z., Quan, Y., Liu, L., Ding, Y., and Xie, Y. 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*, pp. 369–379, 2023.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Fang, C., Zhou, A., and Wang, Z. An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(11):1573–1586, 2022.

Huang, N.-C., Chang, C.-C., Lin, W.-C., Taka, E., Marculescu, D., and Wu, K.-C. ELSA: Exploiting Layer-wise N: M Sparsity for Vision Transformer Acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8006–8015, 2024.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100, 2024.

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

Marcu, A.-M., Chen, L., Hünermann, J., Karnsund, A., Hanotte, B., Chidananda, P., Nair, S., Badrinarayanan, V., Kendall, A., Shotton, J., and Sinavski, O. Lingoqa: Visual question answering for autonomous driving. *arXiv preprint arXiv:2312.14115*, 2023.

Miao, X., Oliaro, G., Cheng, X., Wu, M., Unger, C., and Jia, Z. Flexllm: A system for co-serving large language model inference and parameter-efficient finetuning. *arXiv preprint arXiv:2402.18789*, 2024.

Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., and Gao, J. Large language models: A survey. *arXiv preprint arXiv:2402.06196*, 2024.

Moitra, A., Bhattacharjee, A., Kim, Y., and Panda, P. Trex-reusing vision transformer's attention for efficient xbar-based computing. *arXiv preprint arXiv:2408.12742*, 2024a.

Moitra, A., Bhattacharjee, A., and Panda, P. Pivot-input-aware path selection for energy-efficient vit inference. *arXiv preprint arXiv:2404.15185*, 2024b.

Murthy, R., Yang, L., Tan, J., Awalgaonkar, T. M., Zhou, Y., Heinecke, S., Desai, S., Wu, J., Xu, R., Tan, S., Zhang, J., Liu, Z., Kokane, S., Liu, Z., Zhu, M., Wang, H., Xiong, C., and Savarese, S. Mobileaibench: Benchmarking llms and lmms for on-device use cases, 2024. URL https://arxiv.org/abs/2406.10290.

Park, S.-S., Kim, K., So, J., Jung, J., Lee, J., Woo, K., Kim, N., Lee, Y., Kim, H., Kwon, Y., et al. An lpddr-based cxl-pnm platform for tco-efficient inference of transformer-based large language models. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 970–982. IEEE, 2024.

Qin, Y., Wang, Y., Deng, D., Zhao, Z., Yang, X., Liu, L., Wei, S., Hu, Y., and Yin, S. Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pp. 1–14, 2023.

Qin, Y., Wang, Y., Zhao, Z., Yang, X., Zhou, Y., Wei, S., Hu, Y., and Yin, S. Mecla: Memory-compute-efficient llm accelerator with scaling sub-matrix partition. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 1032–1047. IEEE, 2024.

Tambe, T., Hooper, C., Pentecost, L., Jia, T., Yang, E.-Y., Donato, M., Sanh, V., Whatmough, P., Rush, A. M., Brooks, D., et al. Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 830–844, 2021.

Tambe, T., Zhang, J., Hooper, C., Jia, T., Whatmough, P. N., Zuckerman, J., Dos Santos, M. C., Loscalzo, E. J., Giri, D., Shepard, K., et al. 22.9 a 12nm 18.1 tflops/w sparse transformer processor with entropy-based early

exit, mixed-precision predication and fine-grained power management. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 342–344. IEEE, 2023.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pp. 10347–10357. PMLR, 2021.

Wang, H., Xu, H., Wang, Y., and Han, Y. Cta: Hardware-software co-design for compressed token attention mechanism. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 429–441. IEEE, 2023.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Xu, H., Li, Y., and Ji, S. Llamaf: An efficient LLAMA2 architecture accelerator on embedded FPGAs. *arXiv preprint arXiv:2409.11424*, 2024.

Zeng, S., Liu, J., Dai, G., Yang, X., Fu, T., Wang, H., Ma, W., Sun, H., Li, S., Huang, Z., 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*, pp. 223–234, 2024.

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

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhao, Y., Wu, D., and Wang, J. Alisa: Accelerating large language model inference via sparsity-aware kv caching. *arXiv preprint arXiv:2403.17312*, 2024.