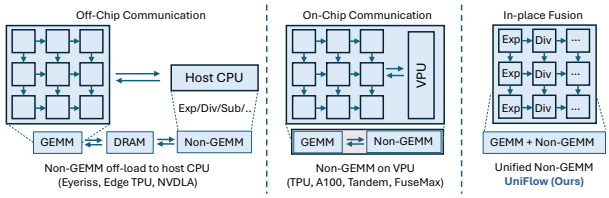
UniFlow: A Spatial Transformer Accelerator with Unified GEMM and Non-GEMM Dataflow Architecture

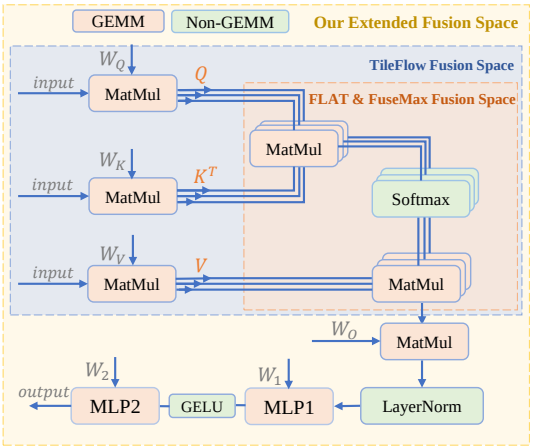
Anonymous Author(s)

Abstract

Spatial accelerators for Transformers face a critical inefficiency: while they excel at GEMM operations through dataflow execution, they must offload non-GEMM operations (Softmax, LayerNorm, GeLU) to vector units or CPUs, breaking the spatial computing paradigm. As non-GEMM operations now account for up to 50% of transformer inference latency, this fragmentation severely limits performance. We present UniFlow, a spatial accelerator architecture that rethinks how Transformer operations should execute on spatial arrays. Our key insight is that non-GEMM operations can be decomposed into dataflow-compatible primitives that execute natively on augmented PE arrays, eliminating the need for separate vector units. UniFlow contributes: (1) an extended fusion space that makes all transformer operations dataflow-decomposable, (2) a PE microarchitecture that enables dynamic per-PE switching between GEMM and non-GEMM execution during runtime with only 14% area overhead, (3) spatial PE array partitioning enabling fine-grained pipeline parallelism between GEMM and non-GEMM operations, and (4) KV cache-aware fusion optimization that co-optimizes memory allocation between caching and operation fusion. Our evaluation shows that UniFlow can boost model inference performance by up to 1.4-1.8× speedup over state-of-the-art non-GEMM fusion works and achieve 1.2-5.2× speedup for the decode stage when considering spatial PE partition and KV Cache.



(a) UniFlow vs. Existing Hardware for Non-GEMM



(b) Major Fusion Space UniFlow vs. Other Work

Figure 1: Existing Support for Non-GEMM Dataflow

1 Introduction

Machine learning workloads are gradually shifting from simple multilayer perceptrons, feed-forward models, and convolutional neural networks (CNNs) to large-scale Transformer-based models [50]. Unlike CNNs, which consist mainly of convolutions, self-attention blocks include multiple matrix multiplication blocks and memory-bound operators (non-GEMM operations, e.g., Softmax), which leads to a lower overall arithmetic intensity compared to CNNs [28]. In addition, Transformer layers involve complex operator dependencies as shown in Figure 1b. Such new characteristics of Transformer workload motivate specialized accelerator architectures for Transformers to exploit new optimization opportunities implied by the specific dependency patterns. Moreover, the recent applications of Transformer, such as document summarization [8], presentation generation [54], and video generation [37], result in a significant increase in memory demands.

As a potential solution for memory bottleneck, researchers have explored operator fusion, which merges sequential operations into a single composite operation and reduces unnecessary off-chip memory access between sequential operators [7, 9, 18, 45]. However, most previous works either focus on GPUs [12, 13] or CNN-based accelerators [7, 18, 57]. Some works target Transformer accelerators, but the operator fusion search space is limited to batch dimension [9] or a subset of operators in Transformer blocks [25, 36, 56].

Some recent studies [19, 26, 41, 44] have suggested the importance of non-GEMM operations in Transformer, including Softmax, LayerNorm, and others. None of the prior work fully remove the limitations for non-GEMM fusion in spatial accelerators due to several reasons: 1) Non-GEMM operations require special hardware support for computation as shown in Figure 1a. Spatial accelerators do not naturally support non-GEMM operations. 2) Without proper hardware support, the existing fusion space, as shown in Figure 1b, is limited. FLAT[25] and TileFlow[56] lack hardware support for non-GEMM operations. FuseMax[36] handles only Softmax, ignoring LayerNorm and GeLU. Tandem Processor[19] supports only two-operator fusion. None of them achieves comprehensive dataflow-level fusion across all transformer operations.

Our insight is that these non-GEMM operators can be decomposed into fine-grained mapping to spatial accelerators with some hardware support to enable in-place fusion, which will further reduce on-chip communication overhead. To support such non-GEMM mapping to spatial accelerators, the traditional PE only contains a MAC is not enough. While commercial spatial chips such as Cerebras CS-2 [32] and SambaNova SN40L [42] place non-GEMM operators on spatial fabric, they do not achieve this level of

fusion for two reasons: (1) each PE is configured for a single operation at runtime, requiring inter-PE communication between every operator stage; and (2) their array topologies lack the reduction-aware connectivity ($Reduce_{Max}$, $Reduce_{Sum}$) that Softmax demands, making true in-place GEMM–Softmax–GEMM dataflow fusion impossible without off-PE coordination. UniFlow co-designs the PE microarchitecture and array topology specifically to eliminate both limitations. With this hardware design, we decompose all non-GEMM operations into dataflow-level primitives, including the original Softmax or more advanced flashattention-style, which is essential when sequence length becomes increasingly longer in recent trends [12, 13, 36].

After extending fusion space for spatial accelerators, we can use latency and energy as metrics to explore the tiling and fusion strategies of the transformer workload. We also investigate how KV cache and PE spatial partitioning affect the fusion scheme for the decode stage of generative LLMs. In summary, our contributions are as follows:

- We extend and clearly define the design space for dataflow fusion on spatial accelerators with non-GEMM operator mapping;
- We propose a Unified GEMM and non-GEMM Dataflow Accelerator, **UniFlow**, with special PE and bus for supporting non-GEMM operators;
- We introduce a fusion flow with KV cache considered and PE spatial partitioning for decoding stage.

2 Background and Motivation

To better understand the need and challenges for accelerating Transformer models with spatial accelerators and current hardware support for dataflow fusion, we elaborate on the details from model, fusion, and hardware perspectives here.

2.1 Spatial Accelerators and Dataflow Mapping

As shown in Figure 4, a spatial accelerator architecture consists of a PE array (processing element), a local scratchpad memory L_1 within each PE, a shared scratchpad memory L_2, and NoCs (network-on-chips) that interconnect the PE array and shared scratchpad memory L_2. PEs serve as computation units within accelerators, containing ALUs (arithmetic logic units) or MAC (multiply-accumulate units) with a local scratchpad L_1. Dataflow is defined as the mechanism to leverage the maximum data reuse opportunity in domain-specific accelerators. Dataflow mapping typical includes parallelization strategy, loop order, and tiling strategy[49].

2.2 Transformer Model and Generative LLM

As illustrated in Figure 1b, the attention blocks and the feed-forward module first project the input sequence X of dimension $D \times L$ (D and L represent the embedding size and sequence length, respectively) by different weight matrices W_Q, W_K, W_V of dimension $D \times D$ to get the so-called *query* Q, *key* K and *value* V matrices of dimension $D \times L$. The matrix Q and the transposed matrix K will produce an activation matrix A of size $L \times L$. The attention score S will then be calculated based on the value $Softmax$ of the activation matrix A. To obtain the output of the attention block, it will multiply the value matrix V with the attention score S, resulting in a matrix O with a dimension of $D \times L$. Finally, to obtain the output Y of the entire attention block, it projects the matrix O by another weight matrix W_O. After the attention block, the intermediate tensor will be passed into two consecutive linear projection layers MLP with a given hidden dimension d_{FFN} plus nonlinear activation layers of GeLU (Gaussian Error Linear Unit).

Generative LLM inference typically consists of two primary stages: the prefill stage, where the LLM processes the input sequence (or prompt) to summarize its context and generate an initial token, and the decode stage, where the LLM uses the previously generated token to predict the next token in an autoregressive manner [6, 40].

2.3 Operator Fusion and KV Cache

Operator fusion has been widely used to avoid writing intermediate results back to off-chip memory to alleviate the bottleneck introduced by Transformer workloads[25, 56]. However, operator fusion in transformer models is challenging: 1) it's hard to break the data dependencies involving non-GEMM operations. For example, Softmax requires the full computation of QK^T; 2) GEMM can be easily mapped to 2D PE array while non-GEMM requires more fine-grained operations, like element-wise and reduction operations, which introduce delays. As shown in Table 1, previous fusion work either focus on CNNs or does not fully support non-GEMM operators. FLAT[25] tried to target dataflow accelerators and perform inter-layer fusion; however, it is limited to GEMM-Softmax-GEMM fusion, does not fully explore all the fusion schemes, and omits the Softmax data transfer overhead. TileFlow [56] proposed a tree-based fusion for the self-attention layer; however, it models Softmax with GEMM-like dataflow, which is not realistic in the spatial hardware without proper support.

In the typical transformer decoding phase, KV cache remains in off-chip memory (e.g., DRAM and High-Bandwidth Memory), and there are two ways to load KV cache for inference: 1) stream in KV cache as needed; 2) preload KV cache on-chip. Streaming KV cache is suitable for long sequences and memory-limited accelerators and has been widely studied (e.g., prefetching, quantization [21, 33]). However, it will increase DRAM traffic and latency. To reduce off-chip memory transfer and improve temporal reuse of the data, we consider allocating on-chip memory for KV cache and preload KV cache there. This approach introduces an unexplored trade-off between the amount of available memory after fusion and offchip data transfer cost for KV cache in accelerators. (Note that KV cache has been treated separately from fusion optimization [13, 25, 36, 56].) Other emerging architectures featuring larger on-chip SRAM make the trade-off between KV cache preloading and fusion more relevant [32, 42, 46].

2.4 Non-GEMM Operators Challenges

GEMM operators have been the prime optimization target for ML inference. This led to the development and adoption of advanced GPUs and spatial accelerators tailored for boosting the performance of GEMM computations. Nevertheless, the heavy GEMM acceleration shifted the landscape of the Amdhal's law for transformer workloads. Recent studies highlight the significant increase in the relative execution time of non-GEMM operators [19, 26, 41, 44]. We observe in figure 2 that non-GEMM operators account for 50%

Table 1: Existing Fusion Framework ("✓": support ; "-": no support; "✓*" : limited support).

Name	Target Workload	Fusible Operators			KV Cache Aware	HW Support for Non-GEMM
		GEMM	Softmax	Other Non-GEMM		
LoopTree[20]	CNN	✓	-	-	-	-
STREAM[45]	CNN	✓	-	-	-	-
DeFiNES[35]	CNN	✓	-	-	-	-
Chimera[57]	Attention/CNN	✓	✓*	-	-	-
TileFlow[56]	Attention/CNN	✓	✓*	-	-	-
FLAT[25]	Attention	✓	✓*	-	-	-
FuseMax[36]	Attention	✓	✓	-	-	✓*
UniFlow (Ours)	Attention	✓	✓	✓	✓	✓

Table 2: The perplexity scores on Wikitext2 of 3 variants of Llama3 before and after applying HW-friendly softmax approximation from [44].

Model	Full Precision (FP32)	Approximate (INT16)
Llama3 1B	9.91	10.84
Llama3 3B	7.89	8.03
Llama3 8B	6.35	7.94

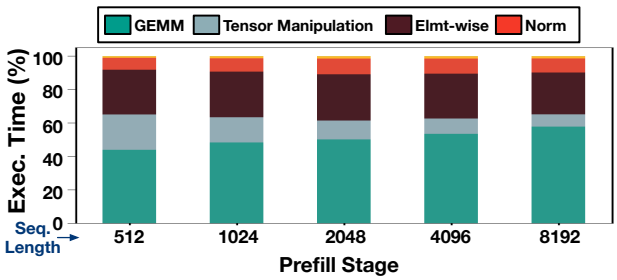


Figure 2: The GPU latency breakdown of the prefill stage of Llama3 8B into GEMM and non-GEMM operators. We apply llm.int8() [14] and run the model on an A100 GPU [4].

of the prefill stage latency of Llama3 [17, 48] running on an A100 GPU. Therefore, it is becoming essential to consider non-GEMM operators in the design space of spatial accelerators and dataflow mapping to ensure optimal execution of transformer inference.

Table 4 shows how existing spatial accelerators support non-GEMM operators. Eyeriss[11], Edge TPU[43], and NVDLA [1] were designed to accelerate GEMM operators and lack of native support for non-GEMM operators. They have to offload non-GEMM to the host CPUs. Google TPU[23], Nvidia GPUs[4], Groq[5], Tesla Dojo[46], Tandem Processor[19], and FuseMax[36] integrate a vector processor unit (VPU) to handle non-GEMM workloads. Cerebras CS-2 [28] and SambaNova SN40L [29] employ fully spatial architectures, mapping non-GEMM operators directly onto their PEs. However, neither achieves dataflow-level in-place fusion for three concrete reasons. First, PE reconfigurability: CS-2 can execute element-wise operations, but the computation is distributed spatially across its processor mesh — each PE holds a fixed role — rather than being fused within a deep temporal pipeline within the same PE array. SN40L maps computation graphs onto its PEs, but each PE is generally configured to perform one specific operation at runtime, making GEMM and non-GEMM stages execute as separate passes with inter-stage communication overhead. Second, reduction topology: Softmax requires *Reduce_Max* and *Reduce_Sum* across an entire row of activations. Neither CS-2's mesh nor SN40L's PE array provides a dedicated intra-column reduction bus; these operations must be resolved outside the normal dataflow, breaking in-place fusion. Third, intra-array partitioning: neither design supports column-granular spatial partitioning that allows GEMM and non-GEMM to execute concurrently and pipeline across tiles within a single PE array. Consequently, neither system achieves the dataflow-level, fine-grained non-GEMM fusion defined in Table III. Furthermore, both designs rely on massive memory subsystems (e.g., the 40 GB of on-wafer SRAM in the CS-2) to service their large PE arrays.

Some recent studies enable lossless 8-bit weight and activation quantization for LLMs [27, 51] which further expands the usage of integer-only spatial accelerators in transformer models. In addition, recent approximations of non-GEMM kernels showed negligible errors [27, 44]. Table 2 shows that applying a piece-wise linear approximation of softmax [44] will not result in significant perplexity degradation. These approximations, facilitate the integration of much needed non-GEMM support in spatial accelerators.The difficulty of fusing and tiling non-GEMM operators has also been addressed by the recent GPU work FlashAttention [12, 13].

Furthermore, some recent work try to provide hardware support for Softmax operations at operator-level and try to provide better fusion options for non-GEMM[19, 36]. However, they still require non-GEMM to be executed on VPUs and overlook the other non-GEMM like Normalization and Activations. No prior work removes all the limitations on the non-GEMM fusion exploration of spatial accelerators.

3 Unified Dataflow Fusion

Having established the limitations of current approaches, we now present our unified dataflow solution: to enable non-GEMM fusion on spatial accelerators. We first need to understand the difference among different fusion, then extend the existing fusion space and make all non-GEMM operators decomposable to dataflow-level to enable unified execution on spatial accelerators.

3.1 Fusion Taxonomy

As shown in table 3, we establish a taxonomy of fusion based on granularity, hardware support, communication overhead, and abstract level. Graph-level fusion operates at the high-level computational graph and relies on software frameworks such as TensorRT[2] and PyTorch[39] for execution, which introduces high communication overhead due to inter-layer dependencies. Kernel-level fusion,

Table 3: Taxonomy of Fusion. UniFlow enables dataflow-level fusion on spatial accelerators.

Fusion Type	Abstraction Level	Granularity	Hardware Support	Communication Overhead	Examples
Graph	Programming	High-level Graph	No dedicated hardware support, relies on software frameworks	High due to inter-layer dependencies	TensorRT[2], PyTorch[39]
Kernel	Programming	Low-level Kernel	Optimized for accelerators	Low as execution consider hardware architecture	FlashAttention[13]
Operator	Architecture	ML Operators (e.g., GEMM + Softmax)	GEMM+GEMM: no additional hardware needed; GEMM+non-GEMM: Use both PE array and VPU for fusion	GEMM+GEMM: no communication overhead; GEMM+non-GEMM: on-chip level communication needed, from PE to VPU	FLAT [25], FuseMax [36], TileFlow [56]
Dataflow	Architecture	Basic Primitives (e.g., matmul, exp, div)	GEMM+non-GEMM: in-place fusion enabled with special PE or PE-to-PE communication)	There is no on-chip level communication as primitives are fused at the lowest level	**UniFlow(Ours)**

Table 4: Non-GEMM Support in Existing Spatial Accelerators

Non-GEMM Support	Accelerators
Host CPU	Eyeriss[11],Edge TPU[43],NVDLA[1]
Vector Processor	TPU[23],A100[4],Groq[5],Dojo[46],Tandem [19],FuseMax[36]
Special PE	Cerebras[32],SambaNova[42],**UniFlow(Ours)**

on the other hand, operates at a lower level of abstraction and is optimized for accelerators. For example, FlashAttention[13] leverages hardware-aware execution to minimize communication overhead and improve efficiency. Operator-level fusion focuses on merging machine learning operators, such as GEMM fused with Softmax or LayerNorm. For the GEMM+non-GEMM, this level of fusion requires both PE arrays and VPUs for efficient execution, which introduces on-chip communication overhead between PE arrays and VPUs for non-GEMM fusion. At the lowest level, dataflow fusion integrates basic primitives (e.g., matrix multiplication, exp, and division) at the hardware execution level, achieving in-place fusion with minimal communication overhead. This type of fusion is directly implemented within spatial accelerators, leveraging specialized PE designs or PE-to-PE communication. By reducing the need for off-chip memory access and minimizing on-chip communication, dataflow-level fusion provides the most efficient approach for executing non-GEMM operations.

3.2 Extended Fusion Space

While prior works [25, 36, 56] explored some non-GEMM operations as shown in Figure 1b, they were mostly limited to Softmax operators. Most studies have overlooked critical computations like exponential functions, normalization, and activation functions. Prior works either ignored the mapping of exponential functions, preventing full exploration of the fusion dataflow space, or they used GEMM-like operations to approximate non-GEMM operations, which introduces some errors in latency evaluation [25, 36, 56]

Figure 5 shows the latency improvement in the LLama3-8B model with different sequence lengths from 128, 512 to 1024. Compared to no fusion, fusion with Softmax can lead up to 1.16× speedup. Compare to no fusion and fusion with Softmax, fusion with LayerNorm and GeLU can lead to 1.44× to 1.84× speedup. This shows the importance of extending our fusion space to support all the non-GEMM operations.

3.3 Dataflow Decomposition for Non-GEMM Operations

To enable non-GEMM operations on spatial accelerators, we decompose each operation into primitive dataflow-level operations. Our key insight is that complex operations like Softmax can be expressed as sequences of simple primitives (max, broadcast, exponential, sum, division) that map naturally to augmented PEs.

We define an operation as dataflow-native if it satisfies three properties: (1) computation using only local PE operations and nearest-neighbor communication, (2) unidirectional data flow without backpressure, and (3) streaming computation without buffering. With this, we can demonstrate that all Transformer non-GEMM operations are dataflow-decomposable on UniFlow, our spatial accelerator that supports non-GEMM within the PE. Architecture details can be found in section 4.

Figure 3 shows the decomposition of GEMM-Softmax-GEMM on UniFlow: data flows top → bottom or left → right; each PE only needs inputs from neighbors; each stage processes data as it arrives. LayerNorm and GeLU follow similar pattern when decomposed into dataflow-level primitives. LayerNorm is decomposed into: 1) Compute mean (Reduce_Sum, element-wise division); 2) Broadcast and subtraction; 3) Variance: Square each element and perform mean computation again; 4) Broadcast and division to get the final output. Scale and shift can be applied using broadcast multiplication and broadcast addition. For GeLU, we adapted the I-Bert[27] version. The decomposition of GeLU includes abs, clip, and sgn, which are realized by the hardware component we propose in the section 4. For sqrt approximation, we adapt a 2nd-degree polynomial function in LayerNorm and GeLU.

For element-wise operations, with PE design in the section 4, we can easily map element-wise operations, including addition, subtraction, multiplication, and division, onto the spatial accelerators, which are controlled by op_code. For Reduce_Max, as shown in Figure 4, we have a max selector that compares the current maximum value with the value that resides in the PE and propagates through the $output_down$ port. At the last row of the accelerator, the maximum value will be broadcast back to each PE via the vertical bus. For Reduce_Sum, similarly, the summation will be accumulated in each PE and broadcast back to each PE at the last row of 2D PE array.

For exponential calculation, we adapted I-Bert [27] exp approximation, where it can be decomposed into a 2nd-degree polynomial function $L(p)$ and a z bit shift operation

$$exp(x) = L(p) >> z, \tag{1}$$

where p and z that correspond to input x can be found through a look-up table. Thus, We get e^x by shifting $L(p)$ z bits to the right, where $L(p)$ is the 2^{nd}-degree polynomial function. Original $L(p)$ has floating-point coefficients. Since our accelerators are integer only, we multiply each coefficient in $L(p)$ by 1024 to make the

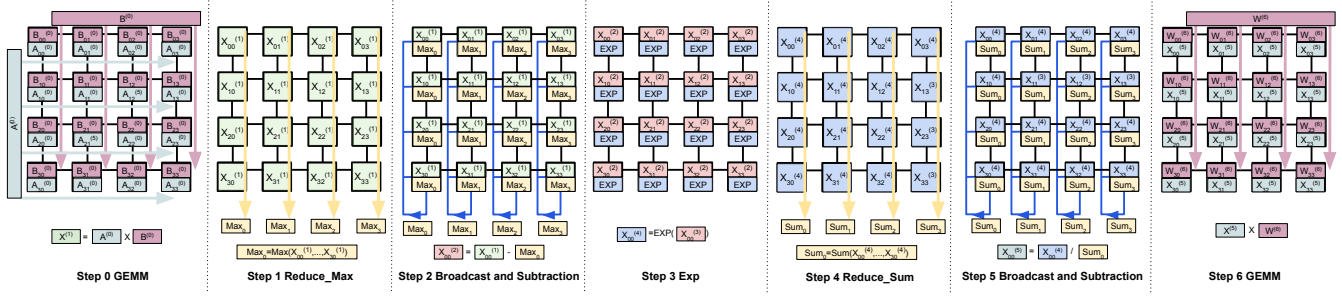


Figure 3: Dataflow of GEMM-Softmax-GEMM within UniFlow.

coefficients integer, which is equivalent to a left shift by 10 bits:

$$T(p) = L(p) \times 1024 = 367p^2 + 993p + 1024 \tag{2}$$

After getting $T(p)$, we just need to shift right by $(10 + z)$ bits to get the approximation of exponential functions

$$exp(x) = T(p) >> (10 + z) \tag{3}$$

The approximation error for exp is 1.9×10^{-3}, even smaller than the quantization error introduced by 8-bit quantization ($1/256 = 3/9 \times 10^{-3}$), which is negligible. GeLU approximation achieves comparable or slightly better results to the non-approximated full-precision GeLU when using RoBERTa-Large Model[34] as baseline along with the QNL1, SST-2, MPRC, and RTE tasks.[27].

With above-mentioned primitives, all non-GEMM operations now can be decomposed into dataflow-level mapping and mapped onto spatial accelerator for in-place fusion.

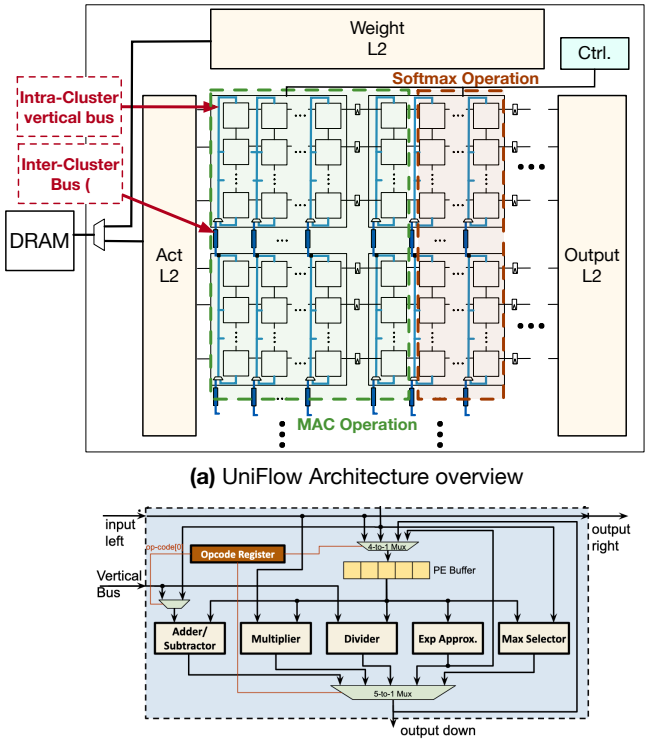
4 UniFlow Architecture Design

With the non-GEMM dataflow decomposition defined, we next describe the hardware architecture that implements these operations. To support both GEMM and non-GEMM dataflow-level fusion on spatial accelerators, we make substantial changes to the accelerator design to support all non-GEMM dataflow-level primitives.

4.1 Overall Architecture

As shown in Figure 4, UniFlow, is a spatial accelerator designed to efficiently fuse both GEMM and non-GEMM operators within the same PE array. UniFlow consists of L_2 buffers, L_1 scratchpad, multiple PE array tiles, and a controller to manage the dataflow execution. Each PE is reconfigurable to support MAC for GEMM execution, approximate non-GEMM operators including Softmax and other non-GEMM operations(as discussed in 3.3). Uniflow has a hierarchical structure with multiple PE array clusters and can be scaled easily. Each cluster can perform an operation different to other clusters. A cluster is comprised of a square-shaped PE array connected to each other in the systolic fashion. Within each cluster, each column can also operate separately from the other columns. Figure 4 depicts an example of Accelerator partitioning for GEMM and non-GEMM operations.

A key enabler for flexible fusion dataflow and accelerated non-GEMM operation is the hierarchical vertical bus. It enables instant process of operators like Reduce_Sum and efficient PE-PE communication, which makes in-place fusion feasible. Each column bus in the cluster is a single-cycle connection between all the PEs in one



(a) UniFlow Architecture overview

(b) Processing Element (PE) micro-architecture

Figure 4: Proposed Spatial Accelerator Architecture: UniFlow

column. Clusters in the same x dimension are connected to each other by multiple bridge buses as demonstrated in 4, making all the PEs in the same column across the accelerator connected to a hierarchical interconnection. Using the systolic dataflow and the broadcast bus, the process of operations like reduce_max, broadcast, and distribution will be accelerated significantly($o(1)$).

UniFlow enables spatial partitioning of the PE columns and prepares required infrastructure for the custom fusion dataflow. Compute units can be partitioned into two parts for GEMM and non-GEMM operations. For instance, a GEMM operation is fuesd with a Softmax on UniFlow. Offline analysis determines and assigns the required compute resources for each operation to balance the throughput. Systolic interconnects alongside vertical buses enable dataflow from the data produced in "GEMM" part and deliver it to the "Softmax" part for consumption without involving memory.

4.2 Processing Element Design

PEs in UniFlow are designed to support approximate non-GEMM operations as well as conventional GEMMs without including complicated operators such as precise exponential. Alongside mesh-style connections which is typical in systolic arrays, each PE can communicate with the bottom and top PEs utilizing the vertical bus. Each Processing Element (PE) in our design supports addition, subtraction, multiplication, division, comparison, and exponent approximation. Crucially, each PE selects its active function via an op_code register, allowing the same physical PE to act as a MAC unit for GEMM in one cycle and as an ALU primitive (e.g., exp-approx, max-selector, divider) for non-GEMM in the next—all within a single uninterrupted dataflow pass. This dynamic per-PE reconfigurability is the key microarchitectural distinction from designs like SambaNova SN40L, where each PE is committed to one operation class at deployment time, and from Cerebras CS-2, where element-wise non-GEMM is distributed across a static spatial mesh without temporal fusion. The exponent approximation procedure uses a small lookup table (LUT) for p and z values based on input [27], and incorporates a divider, adder, and multiplier to generate the e^x. LUT size is a design choice with a trade-off: The larger the LUT, the more accurate the approximation at the cost of more power and area. We chose a 16-row LUT since it provides enough accuracy while not imposing large hardware overhead.

4.3 PE Array and Control Signal Design

Our PE array follows the general systolic array structure with some modifications. First, each column has a dedicated bus which is used for *max* value and *sum* distribution, which is crucial for non-GEMM operations. This hierarchical vertical bus—intra-cluster single-cycle connections plus inter-cluster bridge buses—provides the reduction topology that Softmax requires (*Reduce_Max* followed by broadcast, then *Reduce_Sum* followed by broadcast and division). Commercial spatial chips that lack this dedicated column-reduction topology, such as Cerebras CS-2, cannot implement these reduction steps natively within the PE array and must resolve them through separate coordination paths, breaking in-place dataflow fusion. The lowest PE is the data producer, and the rest of the PEs are consumers. In the Softmax calculation, the maximum value is broadcast from the bottom to all other PEs. If the vector is larger than the PE numbers in a column, the bus is used as a torus connection to bring the current maximum value back to the top PE, so the *max* operation continues on newly brought values until the vector is finished. Second, each PE receives an *op_code* to specify which module and datapath should be activated in the current operation mapping. Another contribution of this work is to enable PE array partitioning, so they can operate on different operations in the compute graph. This enables a wide flexibility for finding an ideal fusion and mapping.

The granularity of PE array partitioning is the column. As you can see in the example in Figure 4, some columns are dedicated to the Softmax operation, while the rest of the PE array is exploited for the GEMM operation. The control signal is broadcast along with a column ID to all PEs for minimum area and power overhead. Each PE checks its column ID and takes the *op_code* if it matches. It takes *Num_PE_X* cycles for the entire accelerator to have the *op_code*, which is negligible according to the partitioning change frequency. The rest of the accelerator components are the same as the systolic array[19, 22].

5 UniFlow Dataflow Mapping

A unified PE that supports both GEMM and non-GEMM primitives expands the design space the mapper must search: the same physical resources can now be allocated to GEMM, to non-GEMM, or to both in different phases of execution. This section describes how UniFlow makes that choice. We first state the optimization problem (Section 5.1), then describe the three execution modes the mapper chooses among (Section 5.2), introduce the notation needed to express a mapping (Section 5.3), and finally specify the search procedure (Section 5.5).

5.1 The Mapping Problem

Given a transformer layer (or a stack of d fused layers) and a UniFlow accelerator with a $P \times P$ PE array, M_{avail} of on-chip memory, and DRAM bandwidth BW_{DRAM}, the mapper selects (i) a tiling of every operator at each memory level, (ii) a fusion strategy that determines which operators execute in a shared pipeline, (iii) a column-wise PE partition ratio α between GEMM and non-GEMM regions, and (iv) for the decode stage, a KV-cache retention ratio β. The objective is to minimize end-to-end latency subject to the on-chip memory constraint:

$$M_{\text{wt}} + M_{\text{act}} + M_{\text{kv}}(\beta) + M_{\text{im}}(d, \text{cut}) \;\le\; M_{\text{avail}}, \tag{4}$$

where M_{wt}, M_{act}, and M_{kv} are the weight, activation, and KV-cache footprints, and $M_{\text{im}}(d, \text{cut})$ is the live intermediate-tensor footprint of the fused subgraph – a quantity that depends both on the fusion depth d and on which producer-consumer edges in that subgraph are *cut* (i.e., materialized to scratchpad rather than streamed through PE-local registers).

We want to highlight one subtlety that prior work has glossed over. The intermediate footprint M_{im} is *not* a function of d alone: a fused subgraph with deeply pipelined producer-consumer pairs requires only the live tile of each pair to be resident, regardless of how many operators are fused above it. The term grows with d only when the mapper cannot find a streaming schedule that retires intermediates as fast as they are produced – which happens precisely when on-chip primitives saturate, when column-bus broadcasts serialize, or when the KV cache for a fused layer cannot fit alongside the working set of the layers above it. The trade-off in the decode stage, then, is not simply "deeper fusion costs memory"; it is that increasing d shrinks the slack available for M_{kv} in Eq. 4, and once β drops below 1 the streamed portion of the cache costs DRAM bandwidth on every decode step. We characterize this trade-off empirically in Section 6.4.

5.2 Three Execution Modes

UniFlow's mapper selects among three modes for any pair of adjacent operators in the transformer layer:

Sequential – each operator runs on the full PE array, intermediates spill to scratchpad or DRAM. This models the offload-class baseline (Eyeriss [11], NVDLA [1]) when non-GEMM is shipped to host.

Pipelined – adjacent operators run in distinct time phases on the same PEs, with intermediates buffered locally. This is the mode shown in Figure 3: the GEMM-Softmax-GEMM sequence executes in seven steps (steps 0–6) on the same PE cluster, with the column-wise vertical bus carrying the `Reduce_Max` and `Reduce_Sum` results between phases. Pipelined mode maximizes per-PE utilization but serializes GEMM and non-GEMM in time.

Spatially-partitioned – The column-granular partitioning is enabled precisely by the PE *op_code* reconfigurability introduced in Section IV.B. In architectures where PEs are statically assigned to one operation type [42], any partitioning must be decided at design time and cannot be tuned per workload or per sequence length. UniFlow's dynamic *op_code* control allows the partition ratio α to be set per inference run, which is what makes the mapping space exploration in Section V.E tractable. The column dimension of the PE array is split between a GEMM region of width $\lfloor \alpha P \rfloor$ and a non-GEMM region of width $\lceil (1-\alpha)P \rceil$, executing concurrently on adjacent tiles of the same operator pair. This trades per-PE utilization for inter-operator parallelism. The optimal α balances the two regions:

$$\alpha^* = \arg\min_{\alpha} \max\left(\frac{T_{\text{gemm}}}{\alpha}, \frac{T_{\text{ngemm}}}{1-\alpha} \right), \tag{5}$$

where T_{gemm} and T_{ngemm} are the unpartitioned critical-path latencies of the two operator classes for the current tile.

Pipelined and Spatially-partitioned are not exclusive. The mapper may pipeline a GEMM-Softmax pair while spatially partitioning the surrounding GEMM-GEMM-Softmax-GEMM-GEMM block; the choice is per operator-pair within a fused subgraph. In Figure 3 we illustrate Pipelined mode for clarity; Section 6.5 reports results when both modes coexist within a single fused subgraph.

5.3 Mapping Notation

To describe a complete mapping concisely we use a tile-and-mode notation:

$$U_{L_i}^{\text{op}} \left[D_1^{f_1}, D_2^{f_2}, \ldots, D_n^{f_n}; \alpha \right], \tag{6}$$

where op identifies the operator (e.g., $QK^\top$, Softmax, AttnV), $L_i \in \{L_1, L_2\}$ names the memory level at which the tile is held, $D_j^{f_j}$ gives the size f_j of the tile along dimension D_j (batch B, heads H, sequence L, head dim K), and α is the column partition ratio for that operator. For example, attention-score computation held at L_1 on a tile of size (b_2, h_2, l_2, k_2) at partition ratio 1 (full GEMM) is written

$$U_{L_1}^{QK^\top} \left[B^{b_2}, H^{h_2}, L^{l_2}, K^{k_2}; 1 \right]. \tag{7}$$

We compose mappings across the operator pair using mode operators Seq, Pipe, Par that correspond to the three execution modes above. UniFlow's fused softmax is then:

$$\textsc{Pipe}\Big(\textsc{Par}\big(U_{L_1}^{\max}, U_{L_1}^{\exp}, U_{L_1}^{\text{sum}}, U_{L_1}^{\div}\big),\ U_{L_2}^{\textsc{AttnV}}\Big). \tag{8}$$

5.4 KV-Cache-Aware Decode Mapping

For decode we must also decide β, the fraction of the KV cache retained on-chip. With L context tokens, H heads, D head dimension, batch size B, and 2 bytes per element:

$$M_{\text{kv}}(\beta) = 2\,\beta\,B\,L\,H\,D \tag{9}$$

Table 5: Model Configurations

Model	Number of Heads	Sequence Length	Hidden Dimension	Exec. Stage
Bert-Base	12	512	768	prefill
Bert-Large	16	512	1024	prefill
Vit/14-Base	12	256	768	prefill
Vit/14-Huge	16	256	1280	prefill
Vit/16-Base	12	196	768	prefill
Vit/16-Huge	16	196	1280	prefill
Llama3-128	32	128	4096	both
Llama3-512	32	512	4096	both
Llama3-1024	32	1024	4096	both

The streamed remainder $(1-\beta)\,M_{\text{kv}}$ pays BW_{DRAM} on every decode step. The mapper picks β jointly with d subject to Eq. 4, choosing the largest β that leaves enough slack for the live intermediate footprint of the fused depth-d subgraph. Section 6.4 reports the (d, β) pairs UniFlow's mapper selects for the Llama3 configurations and the resulting decode-latency benefit.

5.5 Search Procedure

For intra-layer tiling we use a genetic algorithm with 1,000 trials per operator on TileFlow [56] and Timeloop [38], optimizing for end-to-end layer latency under Eq. 4. For inter-layer fusion we enumerate the small space of Seq/Pipe/Par compositions for the operator-adjacency graph of the transformer layer (Figure 1b), evaluate each composition under the best intra-layer tiling found by the genetic algorithm, and select the latency-optimal pairing. The PE partition ratio α defaults to 0.75 (75% GEMM, 25% non-GEMM) and is tuned per-workload in $\{0.375, 0.5, 0.625, 0.75\}$; Section 6.5 reports the sensitivity of decode latency to α. The KV retention ratio β is selected by binary search over $[0, 1]$ given the fixed d and tiling.

6 Evaluation

6.1 Experiment Settings

We evaluate UniFlow using two hardware configurations (Table 6): an edge device with 32×32 PEs and 4MB L1 cache, and a cloud setup with 256×256 PEs plus 20MB L1 and 40MB L2. We evaluate the transformer models used by FLAT[25], TileFlow[56] and FuseMax[36]: Bert-family [15], ViT-family[16]. Details of model configuration of number of attention heads, sequence length and hidden dimension can be found in the table 5. In addition to the prefill-only architecture, we also model the decode stage of the most recent generative transformer model Llama3-8B[17, 48]. For hardware, we design the different scales of our accelerator in RTL-Level with SystemVerilog and synthesize the design with Synopsys Design Compiler using 15nm technology node library to model the area and power consumption of our design for various scales.

For CPU setting, we assume using AVI-512 running at 2.5 GHz, which processes 16 floats per cycle. For data transfer, we assume PCIe bandwidth is 32 GB/s.

6.2 Non-GEMM Fusion Results

Figure 5 shows the latency improvement from offloading non-GEMM operations to the CPUs in the LLama3-8B model with different sequence lengths from 128, 512 to 1024. TileFlow[56] (equals

Table 6: Hardware Configurations

Config	# of PEs	SRAM Size	SRAM Bandwidth	DRAM Bandwidth
Edge	32 × 32	L1:4 MB	500 GB/s	25 GB/s
Cloud	256 × 256	L1:20 MB L2: 40MB	L1: 4000 GB/s L2: 800 GB/s	160 GB/s

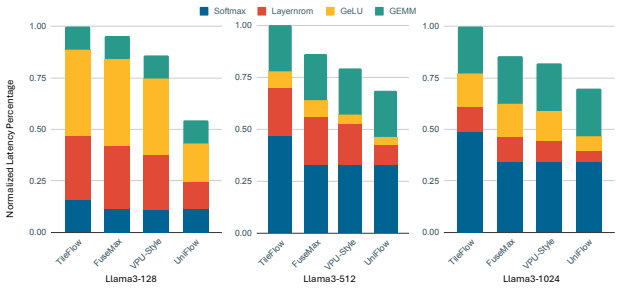


Figure 5: Non-GEMM Fusion. TielFlow: all the non-GEMM operators are executed on host CPU; FuseMax: Softmax has been fused and executed by a VPU, and the rest non-GEMM operators are executed on host CPU; FuseMax Plus (Tandem [19] or TPU[23]-like): all the non-GEMM operators are executed by a VPU; UniFlow: all the non-GEMM operators are fused within the same PE array.

to no Non-GEMM Fusion) offloads all the non-GEMM kernels, while FuseMax[36] only offloads LayerNorm and GeLU because it has hardware support for Softmax. FuseMax Plus is equivalent to Tandem[19] or TPU[23]. Our UniFlow achieves 1.2-1.6× speedup against VPU_Style, 1.3-1.7× speed up against FuseMax and 1.4-1.8× speed up against TileFlow.

6.3 Full Fusion Comparison

Figure 6 shows how UniFlow performs against existing fusion frameworks. With the above setting, we implement 8 different fusions for transformer models and each model has been searched via a genetic algorithm to find the optimal tiling factors. **No_Fuse** execute all the operators sequentially; **FLAT-BGran**[25] pipelined QK^T GEMM and the Softmax without tiling; **FLAT-HGran**[25] fuse GEMM and Softmax with tiling of head dimension; **FLAT-RGran**[25] use same fusion as **FLAT-HGran**[25] with additional tiling in row dimension; **Chimera**[57], **TileFlow**[56], **FuseMax**[36] and our work use same fusion, where GEMM and Softmax are fused, and tiles all the dimensions.

Figure 6 (a) shows the evaluation results of general operator fusion on the edge accelerator. Y-axis shows the normalized cycle and X-axis lists all the models we profile for. Compared to No_Fuse (here not only non-GEMM is not fused but also the GEMM operators, which means all the operators are executed sequentially), FLAT achieves 1.35-1.48× speedup, Chimera achieves 1.42× speedup, TileFlow achieves 2.78× speedup, and our work achieves 4.68× speedup. This shows that increasing tiling granularity will increase the chance of better fusions.

Figure 6(c) demonstrates the normalized cycles for cloud settings. Compared to No_Fuse, FLAT, Chimera and TileFlow only achieve 1.02× to 1.19× speed up due to less spatial utilization, which means tiling granularity has less impact when we have enough hardware resources. With the cloud setting, FuseMax and our work can still achieve 2.41× and 5.48× speedup, which further proves the importance of having hardware support for non-GEMM operations.

Figure 6(b) and (d) show the normalized energy comparison for edge and cloud setting. Since we set the optimization metric to be latency, thus our work does not perform the best in terms of energy. Compared to edge accelerators, we can see that normalized energy is more irregular in cloud settings because 1) with larger hardware resources, PEs might not be utilized efficiently, which could lead to higher energy cost; 2) with more memory levels, different fusion and tiling factors can have a significant impact on memory bandwidth usage, which could also increase the energy. Another reason is that we use a genetic algorithm optimizing only for latency, so the search may fall into a local minimum

6.4 KV Cache Aware Fusion

As shown in Table 7, we evaluate the decoding phase with different decode lengths for KV Cache-aware fusion on the cloud accelerators. For the decode phase, we assume the largest KV Cache size needed to be staged on the chip, even for the first few tokens when KV Cache size is relatively small. For decode length of 128, KV Cache size will be 32 MB. The cloud setting has combined 60 MB SRAM, which is large enough to cache the entire KV values. We explore the different fusions with different depths, which means the number of layers being fused together. The baseline is a fusion depth of 0, meaning no KV Cache is staged on-chip and must be transferred from off-chip memory at every decoding stage. Compared to baseline, fusion with KV Cache, for example, fusion depth = 1, meaning fusion of the entire transformer layer of Llama3-128 with KV Cache staged on-chip, gives us a 5.17× speedup in terms of decoding latency. With more fusion depth, the benefits will be smaller, from 5.17× to 3.98× because the required intermediate memory size will be increasing, which leaves less space for tiling and fusion, and pipelining multiple layers will introduce additional latency if it does not overlap well.

For the longer prefill stage, Llama3-1024, the fusion of more than 1 layer will increase the latency. For the decode length of 512, the KV Cache size is 128 MB, which is larger than the on-chip SRAM available for cloud settings. We can only cache partial KV values and transfer the rest. After manual exploration, we stage 56 MB of KV Cache on-chip, which leaves 4 MB of memory available for tiling and fusion, and transfer the rest of 72 MB of KV Cache values from off-chip memory every time at the decoding stage. For decode length of 1024, with KV Cache size of 256 MB, we also stage 56 MB of KV Cache on-chip and transfer the rest of 200 MB of KV values for decoding.

As we can see, the longer the decode sequence we have, the less speedup we can get from fusion with KV Cache, this is mainly due

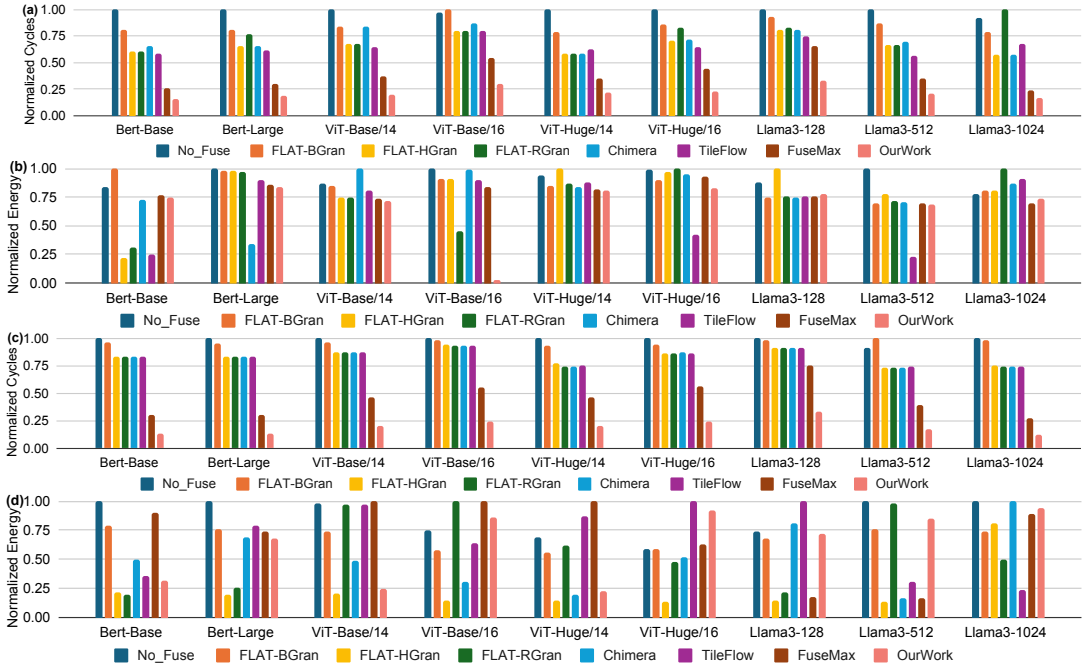


Figure 6: Normalized Latency and Energy on Edge Device (a)(b) and Cloud Device (c)(d).

to the limited space on-chip. With more fusion depth, the trends are similar in all three settings, which gives us fewer benefits in terms of latency.

6.5 Spatial Partitioning

Table 8 shows how PE spatial partition for GEMM and non-GEMM can benefit the latency performance. In the edge setting, we choose the partition to be able to utilize the interleaving between GEMM and non-GEMM operations fully. With different spatial partition ratios, we can get up to 1.6× speedup compared to a baseline without partition.

6.6 Hardware Analysis

We implement the accelerator design proposed in Section 4 in SystemVerilog and synthesize it using Synopsys Design Compiler tools with a 15mn technology library. Table 9 shows the area and power for 8×8, and 16×16, and 64×64 PE arrays. Figure 7b shows the area breakdown on our accelerator design with 16×16 PE array and 1*MB* SRAM. 39.6%, 19.8%, and 19.8% of the area are weight, act, and output SRAM. The PE array occupies 19% of the area, while NoC and controller occupy 1.4% and 0.3% of the area. Compared to TPU Systolic Array, FuseMax has an additional max selector in each PE, while both are equipped with a 1D vector processing unit for non-GEMM operations. UniFlow has an additional divider and a lightweight exponential approximation unit in each PE in the PE array and a vertical bus that connects the PEs. However, it doesn't require 1D vector processor units, since non-GEMM operations are done inside the PE array.

Figure 7a shows the area comparison, with SRAM added, our design compares with FuseMax and TPU Systolic array in 16×16 PE arrays with 1*MB* SRAM, 64×64 PE array with 4*MB* SRAM and 128×128 PE arrays with 8*MB* SRAM. It shows our area has less than 14% and area overhead compared to FuseMax and TPU Systolic Array while achieving 1.68× to 2.26× better latency performance. Tandem processor [19] can be viewed as FuseMax with additional

Table 7: KVCache-Aware Fusion.

Decode Length = 128, KV Cache Size = 32 MB

Fusion Depth	Llama3-128	Llama3-512	Llama3-1024
4	3.984×	2.353×	0.884×
3	4.479×	4.371×	0.91×
2	4.947×	4.831×	0.938×
1	5.179×	5.4×	1.109×
0 (baseline)	1×	1×	1×

Decode Length = 512, KV Cache Size = 128 MB

Fusion Depth	Llama3-128	Llama3-512	Llama3-1024
4	1.695×	1.703×	1.155×
3	1.701×	1.745×	1.165×
2	1.706×	1.75×	1.175×
1	1.708×	1.755×	1.229×
0 (baseline)	1×	1×	1×

Decode Length = 1024, KV Cache Size = 256 MB

Fusion Depth	Llama3-128	Llama3-512	Llama3-1024
4	1.258×	1.255×	1.209×
3	1.259×	1.261×	1.212×
2	1.26×	1.262×	1.214×
1	1.26×	1.263×	1.228×
0 (baseline)	1×	1×	1×

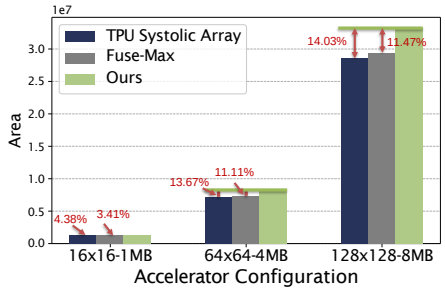
Table 8: PE Spatial Partition for Llama3-128. Baseline means GEMM and non-GEMM are executed sequentially on 32×32 PE array without spatial partition.

GEMM	Non-GEMM	Speedup
32×12	32×20	1.25×
32×16	32×16	1.55×
32×20	32×12	1.60×

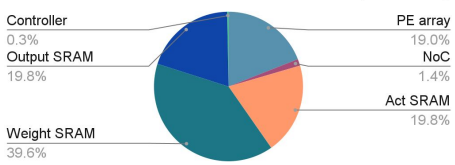
Table 9: Area and Power

PE Config.	Area (mm^2)	Power (mW)
8×8	67221	1.7068
16×16	267154	10.2347
64×64	4288053	202.5778

non-GEMM supports. Here, we think they are equivalent as they are both VPU-based solution to non-GEMM.



(a) Area Comparison of UniFlow vs. FuseMax [36] and TPU systolic array [22].



(b) Area Breakdown of Our Accelerator with 16 × 16 PE Array and 1MB of Global SRAM.

Figure 7: UniFlow Hardware Analysis

7 Related Work

Fusion Space Exploration: At the graph and kernel level: TVM [10] leverages deep learning models to determine efficient fusion strategies;XLA [3], is primarily optimized for CPU and GPU backends supported by LLVM; FlexTensor[58], Ansor[55], and Chimera[57] used scheduler-based fusion optimizations; Diffuse [52] system dynamically performs task and kernel fusion during runtime. These works failed to adapt to transformer fusion space due to a lack of hardware support for non-GEMM operators. At lower level, FLAT[25] tries to target flexible accelerators and perform inter-layer fusion; however, it is limited to activation-to-activation fusion. TileFlow[56], the state-of-the-art fusion dataflow work, fused GEMM and Softmax with indirect modeling of Softmax operations. FuseMax[36], the state-of-the-art Softmax fusion work, proposed to fuse Softmax-only non-GEMM. DeFiNES[35] explored the trade off between fusion depth and tiling size. Cocco[47] provides a recommended memory configuration with graph-level scheduling strategies. Prior work in CGRA[29] fused operators at the coarse-grained level and enabled flexible partitioning of key resources for multi-tasked workloads. Tandem Processor[19] can only interleave GEMM and non-GEMM operations with up to two operator fusions. Compared to them, our work proposed an extended fusion space for transformer workload and thus has a better chance of achieving optimal performance.

Non-GEMM Hardware Support: MAERI [30] focuses on spacial accelerator architectures that employ a tiled general matrix-matrix multiplication (GEMM) kernel. Gamma[24] proposed flexible dataflow accelerators for CNNs workload. I-Bert[27] proposed a hardware-friendly non-GEMM operation. FuseMax[36] and Tandem Processor[19], considered as state-of-the-art hardware support for non-GEMM, propose a 1D PE array and SIMD vector processor for non-GEMM operations. However, they did not utilize the systolic array for reduction types of non-GEMM operations like Reduce_Sum and Reduce_Max, while our architecture supports all types of non-GEMM on spatial accelerators.

Generative LLM Inference: SpliteWise[40] and Sarathi-Serve[6] pointed out the importance of considering both prefill and decode stages for LLM inference. FlightLLM[53] enables these different computation behaviors on FPGAs. InfiniGen[31] used offloading-based LLM serving systems to solve the KV Cache bottleneck. While those are mainly targeting GPUs, very few works in the accelerator domain consider KV Cache in the decoding stage.

8 Limitations and Discussions

While UniFlow enables unified dataflow execution, it may not be the best for workloads with minimal non-GEMM operations due to its area overhead. Future work could explore adaptive PE designs or reconfiguration that dynamically adjust complexity based on workload characteristics. Low PE utilization beyond 64×64 PE array and hard to scale up with hierarchical bus suggest that UniFlow is more cost-effective for edge deployments rather than datacenter-scale implementations. Current UniFlow KV Cache-aware fusion strategies and PE partitioning ratios require manual specification and genetic algorithm search for tiling factors, which is time-consuming and also suggests that a huge design exploration is needed for further optimizations.

9 Conclusion

We extended the existing fusion space in transformer models by including all the non-GEMM operations and providing the corresponding mapping of the non-GEMM. To support non-GEMM, we introduced a spatial accelerator, UniFlow. Lastly, we explore the fusion space with KV Cache considered and spatial PE partition enabled. Compared to the state-of-art work, we can achieve up to 1.8× speedup in the extended fusion space for the prefill stage with hardware support and up to 5.2× speedup for the decode stage compared to the baseline when considering spatial PE partition and KV Cache.

References

[1] [n. d.]. http://nvdla.org/.

[2] [n. d.]. https://github.com/NVIDIA/TensorRT.

[3] [n. d.]. https://openxla.org/xla.

[4] [n. d.]. https://www.nvidia.com/en-us/data-center/a100/.

[5] Dennis Abts, John Kim, Garrin Kimmell, Matthew Boyd, Kris Kang, Sahil Parmar, Andrew Ling, Andrew Bitar, Ibrahim Ahmed, and Jonathan Ross. 2022. The Groq Software-defined Scale-out Tensor Streaming Multiprocessor : From chips-to-systems architectural overview. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. 1–69. doi:10.1109/HCS55958.2022.9895630

[6] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. arXiv:2403.02310 [cs.LG] https://arxiv.org/abs/2403.02310

[7] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.

[8] Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew Gormley. 2023. Unlimiformer: Long-Range Transformers with Unlimited Length Input. In *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 35522–35543. https://proceedings.neurips.cc/paper_files/paper/2023/file/6f9806a5adc72b5b834b27e4c7c0df9b-Paper-Conference.pdf

[9] Jingwei Cai, Yuchen Wei, Zuotong Wu, Sen Peng, and Kaisheng Ma. 2023. Inter-layer Scheduling Space Definition and Exploration for Tiled Accelerators. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–17.

[10] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 578–594.

[11] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138. doi:10.1109/JSSC.2016.2616357

[12] Tri Dao. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. arXiv:2307.08691 [cs.LG]

[13] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. arXiv:2205.14135 [cs.LG]

[14] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. arXiv:2208.07339 [cs.LG]

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *The Ninth International Conference on Learning Representations (ICLR)*.

[17] Abhimanyu Dubey and et. al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[18] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. Tangram: Optimized coarse-grained dataflow for scalable nn accelerators. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 807–820.

[19] Soroush Ghodrati, Sean Kinzer, Hanyang Xu, Rohan Mahapatra, Yoonsung Kim, Byung Hoon Ahn, Dong Kai Wang, Lavanya Karthikeyan, Amir Yazdanbakhsh, Jongse Park, Nam Sung Kim, and Hadi Esmaeilzadeh. 2024. Tandem Processor: Grappling with Emerging Operators in Neural Networks. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 1165–1182. doi:10.1145/3620665.3640365

[20] Michael Gilbert, Yannan Nellie Wu, Angshuman Parashar, Vivienne Sze, and Joel S Emer. 2023. LoopTree: Enabling Exploration of Fused-layer Dataflow Accelerators. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 316–318.

[21] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. arXiv:2401.18079 [cs.LG] https://arxiv.org/abs/2401.18079

[22] Norman P. Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Cliff Young, Xiang Zhou, Zongwei Zhou, and David Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. arXiv:2304.01433 [cs.AR]

[23] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. arXiv:1704.04760 [cs.AR] https://arxiv.org/abs/1704.04760

[24] Sheng-Chun Kao and Tushar Krishna. 2020. GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.

[25] Sheng-Chun Kao, Suvinay Subramanian, Gaurav Agrawal, Amir Yazdanbakhsh, and Tushar Krishna. 2023. 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*. 295–310.

[26] Rachid Karami, Hemanth Kota, Sheng-Chun Kao, and Hyoukjun Kwon. 2024. NonGEMM Bench: Understanding the Performance Horizon of the Latest ML Workloads with NonGEMM Workloads. *arXiv preprint arXiv:2404.11788* (2024).

[27] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. I-BERT: Integer-only BERT Quantization. arXiv:2101.01321 [cs.CL]

[28] Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W Mahoney, et al. 2023. Full stack optimization of transformer inference: a survey. *arXiv preprint arXiv:2302.14017* (2023).

[29] Taeyoung Kong, Kalhan Koul, Priyanka Raina, Mark Horowitz, and Christopher Torng. 2023. Hardware Abstractions and Hardware Mechanisms to Support Multi-Task Execution on Coarse-Grained Reconfigurable Arrays. arXiv:2301.00861 [cs.AR] https://arxiv.org/abs/2301.00861

[30] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems* (Williamsburg, VA, USA) *(ASPLOS '18)*. Association for Computing Machinery, New York, NY, USA, 461–475. doi:10.1145/3173162.3173176

[31] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. arXiv:2406.19707 [cs.LG] https://arxiv.org/abs/2406.19707

[32] Sean Lie. 2022. Cerebras Architecture Deep Dive: First Look Inside the HW/SW Co-Design for Deep Learning : Cerebras Systems. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. 1–34. doi:10.1109/HCS55958.2022.9895479

[33] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. 2024. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. arXiv:2310.07240 [cs.NI] https://arxiv.org/abs/2310.07240

[34] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL] https://arxiv.org/abs/1907.11692

[35] Linyan Mei, Koen Goetschalckx, Arne Symons, and Marian Verhelst. 2024. DeFiNES: Enabling Fast Exploration of the Depth-first Scheduling Space for DNN Accelerators through Analytical Modeling. arXiv:2212.05344 [cs.AR] https://arxiv.org/abs/2212.05344

[36] Nandeeka Nayak, Xinrui Wu, Toluwanimi O. Odemuyiwa, Michael Pellauer, Joel S. Emer, and Christopher W. Fletcher. 2024. FuseMax: Leveraging Extended Einsums to Optimize Attention Accelerator Design. arXiv:2406.10491 [cs.AR] https://arxiv.org/abs/2406.10491

[37] OpenAI. 2024. Video generation models as world simulators. https://openai.com/research/video-generation-models-as-world-simulators. [Online; accessed 04-18-2024].

[38] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. 2019. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 304–315.

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban

Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703 [cs.LG] https://arxiv.org/abs/1912.01703

[40] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient generative LLM inference using phase splitting. arXiv:2311.18677 [cs.AR] https://arxiv.org/abs/2311.18677

[41] Suchita Pati, Shaizeen Aga, Nuwan Jayasena, and Matthew D Sinclair. 2022. Demystifying bert: System design implications. In *2022 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 296–309.

[42] Raghu Prabhakar, Ram Sivaramakrishnan, Darshan Gandhi, Yun Du, Mingran Wang, Xiangyu Song, Kejie Zhang, Tianren Gao, Angela Wang, Xiaoyan Li, Yongning Sheng, Joshua Brot, Denis Sokolov, Apurv Vivek, Calvin Leung, Arjun Sabnis, Jiayu Bai, Tuowen Zhao, Mark Gottscho, David Jackson, Mark Luttrell, Manish K. Shah, Zhengyu Chen, Kaizhao Liang, Swayambhoo Jain, Urmish Thakker, Dawei Huang, Sumti Jairath, Kevin J. Brown, and Kunle Olukotun. 2024. SambaNova SN40L: Scaling the AI Memory Wall with Dataflow and Composition of Experts. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1353–1366. doi:10.1109/micro61859.2024.00100

[43] Kiran Seshadri, Berkin Akin, James Laudon, Ravi Narayanaswami, and Amir Yazdanbakhsh. 2022. An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks. arXiv:2102.10423 [cs.LG] https://arxiv.org/abs/2102.10423

[44] Jacob R Stevens, Rangharajan Venkatesan, Steve Dai, Brucek Khailany, and Anand Raghunathan. 2021. Softermax: Hardware/software co-design of an efficient softmax for transformers. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 469–474.

[45] Arne Symons, Linyan Mei, Steven Colleman, Pouya Houshmand, Sebastian Karl, and Marian Verhelst. 2023. Stream: A Modeling Framework for Fine-grained Layer Fusion on Multi-core DNN Accelerators. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 355–357.

[46] Emil Talpes, Douglas Williams, and Debjit Das Sarma. 2022. DOJO: The Microarchitecture of Tesla's Exa-Scale Computer. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. 1–28. doi:10.1109/HCS55958.2022.9895534

[47] Zhanhong Tan, Zijian Zhu, and Kaisheng Ma. 2024. Cocco: Hardware-Mapping Co-Exploration towards Memory Capacity-Communication Optimization. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 69–84. doi:10.1145/3617232.3624865

[48] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[49] Krishna Tushar, Hyoukjun Kwon, Parashar Angshuman, Pellauer Michael, and Samajdar Ananda. 2020. *Data Orchestration in Deep Learning Accelerators*. Springer Cham.

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[51] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2024. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. arXiv:2211.10438 [cs.CL]

[52] Rohan Yadav, Shiv Sundram, Wonchan Lee, Michael Garland, Michael Bauer, Alex Aiken, and Fredrik Kjolstad. 2024. Composing Distributed Computations Through Task and Kernel Fusion. arXiv:2406.18109 [cs.DC] https://arxiv.org/abs/2406.18109

[53] Shulin Zeng, Jun Liu, Guohao Dai, Xinhao Yang, Tianyu Fu, Hongyi Wang, Wenheng Ma, Hanbo Sun, Shiyao Li, Zixiao Huang, Yadong Dai, Jintao Li, Zehao Wang, Ruoyu Zhang, Kairui Wen, Xuefei Ning, and Yu Wang. 2024. FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGAs. arXiv:2401.03868 [cs.AR] https://arxiv.org/abs/2401.03868

[54] Zhenning Zhang, Yunan Zhang, Suyu Ge, Guangwei Weng, Mridu Narang, Xia Song, and Saurabh Tiwary. 2024. GenSERP: Large Language Models for Whole Page Presentation. arXiv:2402.14301 [cs.IR]

[55] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. 2020. Ansor: generating high-performance tensor programs for deep learning. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI'20)*. USENIX Association, USA, Article 49, 17 pages.

[56] Size Zheng, Siyuan Chen, Siyuan Gao, Liancheng Jia, Guangyu Sun, Runsheng Wang, and Yun Liang. 2023. TileFlow: A Framework for Modeling Fusion Dataflow via Tree-based Analysis. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (Toronto, ON, Canada) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 1271–1288. doi:10.1145/3613424.3623792

[57] Size Zheng, Siyuan Chen, Peidi Song, Renze Chen, Xiuhong Li, Shengen Yan, Dahua Lin, Jingwen Leng, and Yun Liang. 2023. Chimera: An Analytical Optimizing Framework for Effective Compute-intensive Operators Fusion. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1113–1126. doi:10.1109/HPCA56546.2023.10071018

[58] Size Zheng, Yun Liang, Shuo Wang, Renze Chen, and Kaiwen Sheng. 2020. FlexTensor: An Automatic Schedule Exploration and Optimization Framework for Tensor Computation on Heterogeneous System. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 859–873. doi:10.1145/3373376.3378508