Fast Cross-Operator Optimization of Attention Dataflow on Spatial Accelerators

Haodong Chang
ECE Department
Texas A&M University
College Station, Texas, USA
haodong@tamu.edu

Hailiang Hu
AMD Inc
San Jose, CA, USA
hailiang.hu@amd.com

Zhenrui Wang
Texas A&M University
College Station, Texas, USA
zhenruiwang@tamu.edu

Yu Gong
Amazon Web Services
Santa Clara, CA, USA
ygkyle@amazon.com

Rongjian Liang
Nvidia
Austin, TX, USA
rliang@nvidia.com

Zhexiang Tang
Rutgers University
Piscataway, NJ, USA
zhexiang.tang@rutgers.edu

Bo Yuan
Rutgers University
Piscataway, NJ, USA
bo.yuan@soe.rutgers.edu

Jiang Hu
Texas A&M University
College Station, Texas, USA
Technical University of Munich
Munich, Germany
jianghu@tamu.edu

Abstract

Attention is a fundamental computational kernel that accounts for the majority of the workload in transformers and LLMs. Optimizing dataflow is crucial for enhancing both performance and energy efficiency in attention computation. This optimization involves a range of decisions, such as tiling, computation ordering and buffer management, and can be applied at both intra-operator and inter-operator levels, resulting in a highly complex decision space.

We propose a novel approach to cross-operator dataflow optimization. Its centerpiece is an analytical performance model that spans a large decision space and enables matrix-based encoding of multiple candidate solutions. Built on this, a vast number of solutions can be evaluated rapidly, and with an effective pruning technique, the optimal solution can be identified through exhaustive enumeration. We refer to our method as MMEE (Matrix Multiplication Encoded Enumeration). The ability to efficiently enumerate a large design space allows MMEE to deliver higher-quality solutions at a substantially faster speed compared to prior approaches.

The MMEE approach is evaluated across diverse test cases for different accelerator configurations, and consistently achieves the best results with ultrafast speed compared to multiple state-of-the-art methods. For DRAM-traffic-driven optimization, MMEE reduces DRAM traffic by 16%-23%. For energy-driven optimization, MMEE reduces energy by 48%-50% and latency by 31%-69%. For latency-driven optimization, MMEE achieves simultaneous reductions of 40%-50% in energy and 40%-69% in latency. Notably, MMEE is 64× to 287× faster than previous works.

CCS Concepts

• **Computer systems organization** → **Neural networks**.

Keywords

Attention, Fusion, Accelerator



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
PACT '26, Chicago, IL, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2915-7/2026/10
https://doi.org/10.1145/3838684.3846841

1 Introduction

Attention mechanisms play a central role in transformer-based models, which are prevalent across various application domains, including natural language processing [17, 21, 52], computer vision [22, 42], and image generation [47, 70]. As models capture correlations across longer contexts, sequence lengths continue to grow [7, 35, 61]. However, this comes with substantial memory and compute overhead, as attention scales quadratically with sequence length [20, 34, 58, 69]. To address these challenges, numerous techniques have been proposed to improve the efficiency of attention computation on diverse hardware platforms, including CPUs, GPUs, and accelerators [4, 19, 56, 57, 64]. Among these platforms, spatial accelerators offer high energy efficiency and reduced latency due to their specialized hardware architectures.

Dataflow mapping on accelerators dictates how computation and memory resources are utilized spatially and temporally. It plays a critical role in determining the efficiency of individual attention workloads and provides high-fidelity feedback for accelerator designers. Nevertheless, optimizing dataflow mappings remains a complex and challenging problem [48, 73, 74]. Broadly, dataflow optimization can be categorized into intra-operator and inter-operator mappings. Intra-operator mapping focuses on improving the execution of individual operators [24, 32, 41, 50]. However, due to data dependencies across operators, inter-operator mapping, such

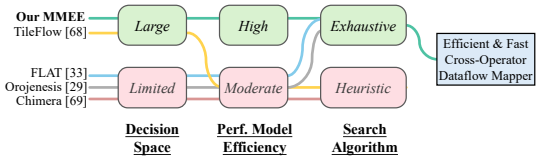


Figure 1: Comparison of cross-operator dataflow mappers.

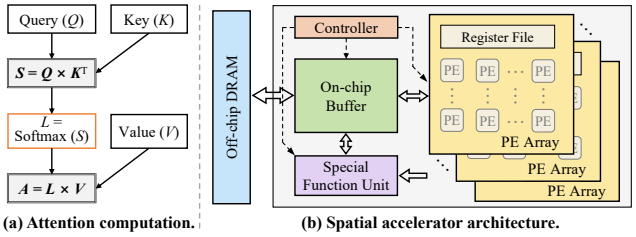


Figure 2: Attention and accelerator architecture.

as operator fusion, offers substantial additional opportunities for improving the overall efficiency of attention computation. The growing importance of inter-operator mappings has spurred recent research into cross-operator techniques that jointly explore both intra- and inter-operator dataflow mappings [9, 33, 73].

In addition to the quality of dataflow mapping solutions, optimization speed is critical in practice. In AI accelerator design, dataflow mapping is a core component repeatedly invoked when evaluating various hardware architectures and model workloads. Similarly, in ML compilers, the dataflow mapper may be frequently executed to accommodate model modifications. Despite its practical significance, the runtime efficiency of cross-operator dataflow optimization has received limited attention in prior research.

How can we obtain high-quality cross-operator dataflow mappings at high speed? We identify three key requirements: (1) *A sufficiently large decision space.* If the decision space is too restricted or omits important options, opportunities for better solutions will be lost. (2) *A fast and accurate performance model.* The model used to estimate latency and energy must be computationally efficient so that many candidate solutions can be evaluated within a reasonable runtime. (3) *A systematic and efficient search algorithm.* Without systematic exploration, high-quality solutions may be overlooked; without efficiency, good solutions cannot be found in practical time. If any of these requirement is not satisfied, a bottleneck is formed to hinder quick finding of high-quality solutions.

In light of these requirements, one can see that existing approaches leave room for improvement. We analyze several state-of-the-art methods and summarize their characteristics with respect to these requirements in Fig. 1. Regarding the decision space, both FLAT [33] and Chimera [74] omit buffer management, while Orojenesis [29] relies on a limited set of templates to constrain the search space. Consequently, even though FLAT and Orojenesis perform exhaustive enumeration, they inherently miss certain solutions due to incomplete decision coverage. All four approaches, FLAT, Chimera, Orojenesis, and TileFlow [73], require parsing of decision scenarios before applying analytical formulas. For instance, TileFlow employs a tree representation, and its model evaluation involves tree traversal. Such parsing significantly slows down performance estimation. TileFlow's search procedure further relies on genetic algorithms and Monte Carlo Tree Search, both of which are randomized heuristics. As a result, they may miss high-quality solutions that exhaustive enumeration would otherwise uncover.

We present a new cross-operator dataflow optimization approach that achieves higher solution quality at much faster speed than prior methods. First, we capture a broad decision space, including tiling, computation ordering, and buffer management. Second, we introduce an analytical performance model without "if–else" branches. This model enables matrix-based encoding of many candidate solutions and efficient evaluation via matrix multiplication. Third, we propose an exhaustive enumeration scheme with pruning that explores a large decision space without sacrificing optimality. We refer to our approach as *Matrix Multiplication Encoded Enumeration* (MMEE). As illustrated in Fig. 1, MMEE is the only one that excels among all three key requirements.

The contributions of our work are summarized below.

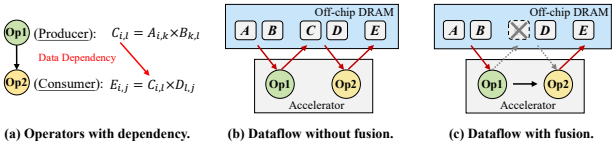
- We demonstrate that, despite the imperfectly nested structure of fusion dataflows, a pseudo nested loop representation provides a systematic and concise abstraction that fully captures the decision space defined by tiling, computation ordering, and buffer management.
- We develop the first analytical model without traversal procedures, to the best of our knowledge, for estimating buffer size requirements, DRAM accesses, energy consumption and latency for fused attention dataflows.
- We enable rapid and optimal dataflow search by combining matrix-based encoding with an effective pruning strategy. This allows exhaustive evaluation of a large design space and ensures the optimal solution can be identified without approximation.

We evaluate MMEE across various workloads on two accelerators. Compared to state-of-the-art methods, MMEE reduces DRAM traffic by 16%-23%. The energy-driven version of MMEE reduces energy consumption by 48%-50% and latency by 31%-69%. For latency-driven optimization, MMEE reduces energy consumption by 40%–50% and latency by 40%–69%. Notably, our optimization process is 64× to 287× faster than prior approaches.

2 Background

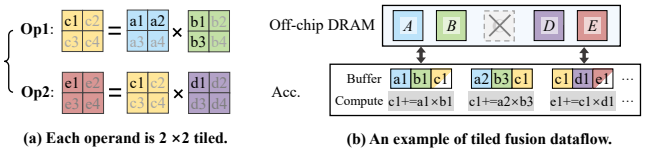
2.1 Attention Computation

The main operations of attention computation are illustrated in Fig. 2(a), where Q, K, V, S, L, and A represent matrices. Each attention head performs the same computation independently. The actual computational load is quite large due to: (1) matrices S and L scale quadratically with sequence length; and (2) frequent invocation of the attention kernel, often exceeding 100 times, due to multiple heads and transformer layers. As shown in [33], the primary benefit of operator fusion arises from combining the matrix multiplications for S and A. However, the presence of softmax operations in between makes the fusion, especially tiled fusion, challenging.



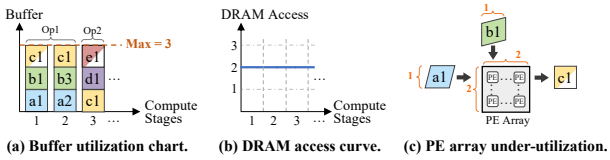
(a) Operators with dependency. (b) Dataflow without fusion. (c) Dataflow with fusion.

Figure 3: Fusion dataflow keeps intermediate results (C) on-chip, avoiding off-chip DRAM accesses.



(a) Each operand is 2 ×2 tiled. (b) An example of tiled fusion dataflow.

Figure 4: Tiling and tiled fusion. (a) Each operand is 2×2 tiled. (b) One tile of intermediate result ($c1$) is kept on-chip instead of C, reducing buffer use from 4 tiles to 1.



(a) Buffer utilization chart. (b) DRAM access curve. (c) PE array under-utilization.

Figure 5: The impact of tiling. Each compute stage represents the multiplication of a pair of tiles. (a) Buffer utilization and (b) DRAM access curves show buffer usage and off-chip traffic with tiling. (c) A small tile size leads to PE under-utilization.

2.2 Accelerator Architecture

Spatial accelerators, also referred to as tiled accelerators, have become increasingly prevalent in accelerator design [9, 11, 30, 73, 77]. As illustrated in Fig. 2(b), a typical spatial accelerator comprises multiple PE arrays, an on-chip buffer, a special function unit (SFU), and a controller [10, 13, 33]. Multiple PE arrays provide inter-PE parallelism while each PE array provides intra-PE parallelism. A single PE array, e.g., systolic array, comprises a 2D array of processing elements (PEs) and associated register files, supporting operations such as matrix multiplication. The on-chip buffer sits between the off-chip DRAM and register files. SFU is responsible for executing specialized functions, such as the softmax operation in attention and layer normalization following feed-forward neural network layers [33, 56]. These components are orchestrated by a dedicated controller to ensure efficient execution.

3 Decision Elements in Fusion Dataflow

In this section, we systematically define fusion and three key decision elements: tiling, computation ordering and buffer management. These elements critically impact buffer capacity, DRAM traffic, and compute utilization. For simplicity, we temporarily omit the softmax operator and focus on the fusion of two matrix multiplications.

3.1 Fusion Dataflow

In Fig. 3(a), the first multiplication acts as the producer operator, generating the intermediate matrix C, while the second serves as the consumer operator, taking C as input. Thus, a data dependency exists between the consumer and producer operators. Fusion represents a dataflow in which the consumer reuses intermediate results without incurring off-chip memory accesses [5, 29, 33], as illustrated in Fig. 3(b) and (c). We abstract three key decision elements that govern such fusion dataflows, detailed in the following sections.

3.2 Tiling

Tiling partitions operands into multiple tiles, a common technique to balance buffer capacity and DRAM access overhead [33, 41, 45, 50]. In fusion dataflow, where intermediate results are often too large to fit in on-chip buffers, tiling can be extended to intermediate outputs, as shown in Fig. 4. Tiled fusion is particularly critical for attention workloads, where the intermediate matrix scales quadratically with sequence length. In Fig. 4(b), applying tiled fusion reduces the buffer demand for storing intermediate results from four tiles to one. Half-filled squares denote tiles with partial sums.

Fig. 5 further illustrates the impact of tiling under the fusion dataflow in Fig. 4. The buffer utilization chart and DRAM access curve visualize the dataflow and its impact on buffer usage and off-chip traffic. In the buffer utilization chart, the horizontal axis denotes compute stages, each corresponding to the multiplication of a tile pair, while the vertical axis represents buffer usage. In the DRAM access curve, the vertical axis indicates transferred to/from DRAM at each stage. If required tiles are not in the on-chip buffer, DRAM accesses occur, e.g., $d1$ is loaded and $e1$ is written back to DRAM at stage 3. Tiling reduces buffer demand to three tiles (Fig. 5(a)), but overly aggressive tiling can yield tile sizes smaller than the PE array (Fig. 5(c)), leading to PE under-utilization.

3.3 Computation Ordering

Unlike intra-operator dataflow, the computation ordering for attention fusion is more complex since it interacts with data dependencies and the softmax operator. Fig. 6 compares two fusion dataflows, where only the left is valid. The right dataflow violates the constraint required for correct attention fusion, summarized below.

Constraint – No Psum Propagation. Intermediate tiles containing partial sums (psums) cannot be forwarded from the producer to the consumer. In the valid dataflow (left), the partial sum of $c1$ produced at stage 1 cannot be consumed at stage 3; only the fully accumulated tile after stage 2 is allowed. However, the right dataflow violates this constraint by consuming $c1$ at stage 2. This restriction follows FlashAttention [18, 46], where each intermediate tile must be fully accumulated before the online softmax. By enforcing this constraint, we can systematically generate any valid attention fusion dataflow without violating producer–consumer dependencies, while ensuring functional correctness.

Recomputation occurs when intermediate results are evicted and later regenerated from the producer. This benefits buffer-limited scenarios by enabling more flexible dataflows and shorter reuse distances, thereby reducing DRAM access. In our framework, recomputation is naturally supported by adjusting inter-tile loop ordering and encoded into the computation-ordering space, as explained in Section 4.2.

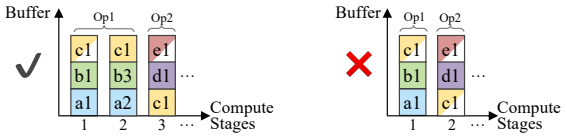


Figure 6: The right computation ordering is invalid, because attention fusion requires $c1$ with complete sums.

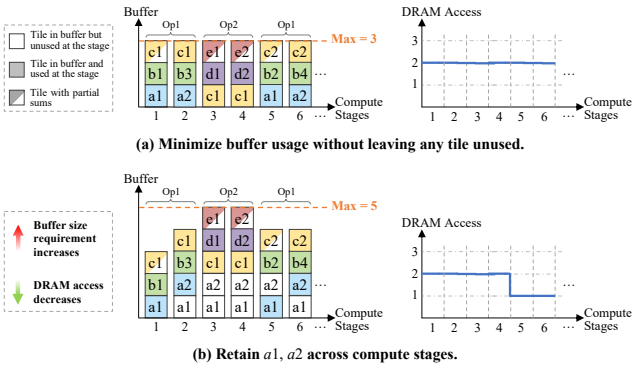


Figure 7: Buffer management and its impact. $a1$ and $a2$ in (b) are retained once loaded into the buffer, eliminating repeated DRAM accesses for $a1$ and $a2$ from stage 5 to stage 6.

3.4 Buffer Management

This decision element concerns how on-chip buffer space is allocated for operands. In intra-operator dataflow mapping, this may be expressed by selecting a buffering level [41, 60], denoting the loop level at which data are buffered for reuse. However, cross-operator dataflow introduces additional complexity. That is, while the accelerator is executing the producer operator, additional buffer space may be allocated to the consumer in order to reduce future DRAM traffic by avoiding repeated loads of consumer operands.

Motivated by this observation, we manage buffer allocation at a fine-grained operand tile level. Specifically, operand tiles may remain in the buffer even when they are not used in certain compute stages, a strategy we refer to as **buffer retention**. In Fig. 7, dataflow (b) retains two tiles of A, $a1$ and $a2$, once they are loaded. As a result, (b) requires more buffer space than (a), but reduces DRAM access.

Crucially, buffer management is tightly coupled with computation ordering. For instance, if we retain $b1$ and $b3$ in Fig. 7(b) without changing the computation ordering, the buffer requirement still increases to five tiles. However, DRAM traffic from stage 5 to stage 6 does not decrease, as tiles $b2$ and $b4$ must still be loaded.

4 Pseudo Nested Loop Representation

We introduce pseudo nested loops as a compact yet expressive representation for tiled fusion dataflows. This form encapsulates all decision elements and, when paired with our performance model, allows exhaustive enumeration of the space. Within this abstraction, loop boundaries, loop order, and buffering levels together define each fusion dataflow in a complete, unique, and concise manner.

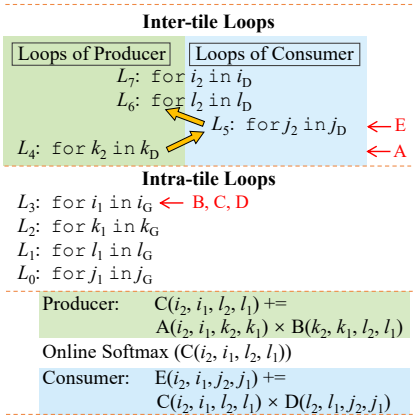


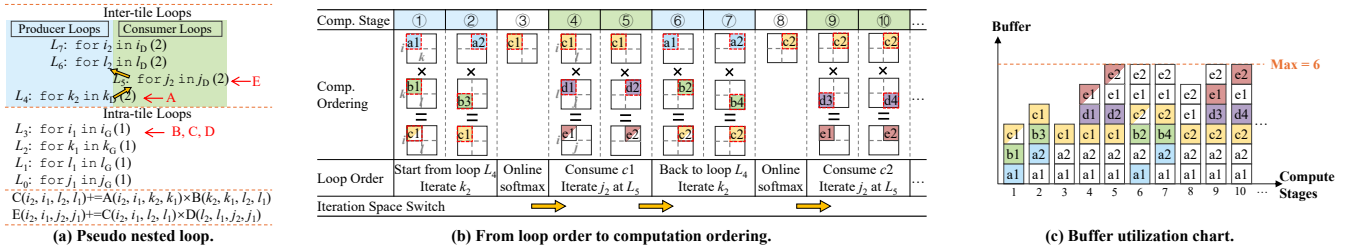
Figure 8: Pseudo nested loop example and components. Loop boundaries come from integer factorization (e.g., $I = i_D \times i_G$). Red arrows mark buffering levels; yellow arrows show transition between iteration spaces.

4.1 Pseudo Nested Loop

A generic pseudo nested loop representation is depicted in Fig. 8, where the outer four loops correspond to inter-tile iterations. Certain loop layers are shared by both the producer and consumer (e.g., L_6 and L_7). The inner four loops correspond to intra-tile iterations.
1) Loop Boundary. Tiling is explicitly captured through loop boundaries, such as i_D, i_G, l_D, and l_G. Each dimension is tiled via integer factorization, e.g., $i_D i_G = I$. These loop boundaries determine operand tile sizes and impact on-chip buffer capacity.
2) Loop Order. Computation ordering is defined by the loop order in the pseudo nested loop. By enforcing the constraint in Section 3.3, the computation ordering of any fusion dataflow, including those with recomputation, can be derived precisely from the inter-tile loop sequence, as shown in Section 4.2.
3) Buffering Level. Buffer management is captured through the buffering level assigned to each operand. Each operand is associated with a specific buffering level, corresponding to one of the loop layers. To represent the buffer-retention behavior described in Section 3.4, we define the following rule: When the buffering level is assigned within inter-tile loops, a buffer space equal to the operand's tile footprint is allocated and retained across all applicable compute stages.

4.2 Learn From an Example

1) Loop Boundary → Tiling. In Fig. 9(a), numeric values are assigned to all loop boundaries. The minimum data unit for every operand is a 1×1 tile, and both data movement and computation operate at this tile granularity. Note that a "tile" here abstracts the actual data volume.
2) Loop Order → Computation Ordering. In Fig. 9(b), unrolling the loop exposes the corresponding computation ordering. We begin with loop layer L_4, where k_2 iterates from 1 to 2, indicating that one tile ($c1$) of the intermediate matrix C is being accumulated. This behavior directly reflects the No Psum Propagation constraint. Then the loop proceeds to L_5, where j_2 marks the iteration to the



(a) Pseudo nested loop. (b) From loop order to computation ordering. (c) Buffer utilization chart.

Figure 9: (a) Numeric values are assigned to loop boundaries to illustrate the tiling decision. (b) Loops are unrolled to reveal the computation ordering. (c) The buffer-utilization chart shows buffer management: buffering levels are assigned to each operand, and the levels for A and E are placed within inter-tile loops, meaning two tiles each of A and E are retained in the buffer.

consumer space. Accordingly, $c1$ is consumed from stage 4 to 5 after online softmax at stage 3. The iteration space transitions from the producer to the consumer are indicated by the yellow arrows at the bottom of the figure.

When the intermediate results of C are not fully consumed, recomputation is applied. For example, after swapping j_2 and l_2 in Fig. 9(a), stages 1–4 remain identical to those in Fig. 9(b). However, at stage 5, the dataflow iterates loop level l_2 in loop layer L_5, which still satisfies the constraint but switches the iteration space back to loop layer L_4. The data tile $c1$ is partially consumed, and buffer usage is reduced by one tile by avoiding storage of $e2$. This swapped case is not shown in Fig. 9(c), but it reduces the maximum buffer usage to 5.

Such iteration-space transitions pose a fundamental challenge for representing fusion, distinguishing it from conventional nested loops [41, 60]. The iteration space later switches back to the producer to generate additional intermediate tiles (e.g., $c2$). Once $c2$ is fully accumulated at stage 8, the consumer processes it from stage 9 to 10, completing another iteration-space switch.

3) Buffering Level → Buffer Management. Fig. 9(c) shows the resulting buffer behavior from the buffering level assignments in Fig. 9(a). Buffering levels for A and E are set inside inter-tile loops, causing buffer retention for their tiles.

For matrix A, the buffering level is set at loop layer L_4, meaning an entire tile row ($a1$ and $a2$) is retained in the buffer once loaded. Consequently, buffer usage increases from 3 to 4 tiles from stage 1 to stage 2, holding $a1$, $a2$, $b3$, and $c1$ at stage 2. Similarly, the buffering level for matrix E is assigned at loop layer L_5, retaining tiles $e1$ and $e2$ once they are loaded. In contrast, matrices B, C, and D set buffering levels at loop layer L_3, discarding one tile once unused. For example, buffer usage decreases from 4 to 3 tiles between stage 2 and stage 3 because $b3$ is discarded after use. When all compute stages finish, the peak buffer usage across stages directly yields the required on-chip buffer size.

5 Analytical Performance Model

In this section, we present a computationally efficient performance model that forms the foundation of MMEE. By leveraging pseudo nested loops (Section 4), we significantly extend an intra-operator model [41] to support cross-operator fusion dataflows. The model is entirely analytical, in contrast to algorithmic models, such as the tree model [73], which requires multiple steps per estimation and thus hinder rapid search over large decision spaces. Although prior DNN performance modeling [71] provides analytical formulas, it relies on numerous "if–else" branches, which increases estimation time. In contrast, our model is fully analytical and avoids any "if–else" parsing. For simplicity, we omit the head dimension in all expressions. This does not sacrifice generality, as computations across different heads are independent [33] and can therefore be easily mapped onto separate PE arrays.

5.1 Terminology

Operand's dimensions. The operand's dimensions refer to the union of its loop variables, e.g., $\{i_2, k_2, i_1, k_1\}$ for matrix A.

Operator's dimensions. The operator's dimensions refer to the union of both operands' dimensions for the operator, e.g., $\{i_2, k_2, l_2, i_1, k_1, l_1\}$ for the producer operator.

Operand's effective dimensions. If there is no recomputation or the operator for the operand is a consumer, the effective dimensions of the operand are the same as the operator's dimensions. For the operand of a producer operator in the presence of recomputation, the effective dimensions are the union of the producer operator's dimensions and the corresponding consumer operator's dimensions. For the example in Fig. 10, if there is recomputation, the effective dimensions of A are $\{i_2, k_2, l_2, i_1, k_1, l_1, j_2, j_1\}$.

Reduction dimension. For an operator, this is the loop variable shared by its two operands, e.g., k_2 for the producer.

Blocker. For an operand, its blocker is the innermost loop layer outside its buffering level whose iteration causes the operand's buffered data to change. For operand A, its buffering level is L_4 in Fig. 10. Its blocker is L_7 because it corresponds to one of operand A's dimensions while L_5 and L_6 have nothing to do with A. For an operand of the consumer, its blocker can also correspond to the reduction dimension of its producer. For example, the blocker of operand D is L_4.

5.2 Buffer Size Requirements Estimation

Operand's buffer size requirement. The buffer size required for an operand is determined by the product of the loop boundaries corresponding to its dimensional loops at and below its buffering level. In Fig. 10, the dimensions for operand matrix A based on its buffering level are k_2, i_1, and k_1, whose associated loop boundaries are k_D, i_G, and k_G, respectively. Thus, the buffer size requirement for A is given by $BS_A = k_D i_G k_G$.

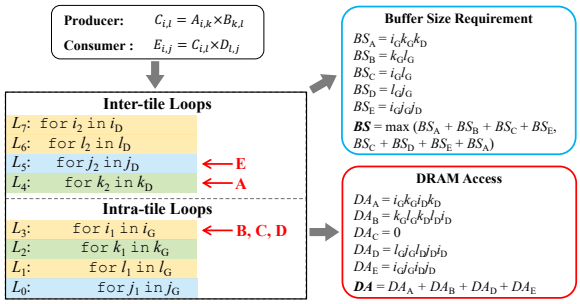


Figure 10: Example of buffer-size requirement and DRAM-access estimation. Each expression is a composition of additive and multiplicative combinations of loop boundaries.

Operator's buffer size requirement. First, we define a binary indicator τ_Y, which is set to 1 if the buffering level of operand Y is at an inter-tile loop, and 0 otherwise. For a given operator M, let Δ^M denote the set of operands exclusively associated with M. In the example shown in Fig. 10, we have $\Delta^{\text{Op1}} = \{A, B\}$. The operand shared between the two operators, Op1 and Op2, is denoted as $\Delta^{\text{Op1,Op2}}$, which in this case is C.

For two fused operators, Op1 and Op2, their buffer size requirements can be estimated as follows:

$$BS^{Op1} = \sum_{X \in \Delta^{Op1} \cup \Delta^{Op1,Op2}} BS_X + \sum_{Y \in \Delta^{Op2}} \tau_Y BS_Y \tag{1}$$

$$BS^{Op2} = \sum_{X \in \Delta^{Op2} \cup \Delta^{Op1,Op2}} BS_X + \sum_{Y \in \Delta^{Op1}} \tau_Y BS_Y \tag{2}$$

Here, $\tau_Y = 1$ indicates that some tiles of operand Y must remain in the buffer even when its associated operator is not currently being executed. For example, in Fig. 10, some tiles of matrix E are buffered during the execution of Op1, even though E is not an operand of Op1. Consequently, the buffer size requirement for Op1 is given by:

$$\begin{aligned} BS^{Op1} &= BS_A + BS_B + BS_C + \tau_D \cdot BS_D + \tau_E \cdot BS_E \\ &= BS_A + BS_B + BS_C + BS_E \end{aligned} \tag{3}$$

This formulation highlights a key difference from the models in [41], which are not designed to account for the complexities introduced by operator fusion.

Overall buffer size requirement. The two fused operators are executed sequentially and therefore the overall buffer size requirement is the maximum between them:

$$BS = \max(BS^{Op1}, BS^{Op2}) \tag{4}$$

5.3 DRAM Access Estimation

DRAM access refers to the volume of data transferred between the on-chip buffer and off-chip DRAM. Intermediate results, such as matrix C, are not permitted to be written to DRAM, and therefore its DRAM access $DA_C = 0$. The derivation of DRAM access in our approach is significantly more complex than that in [41] due to the data dependencies in the fused operators and recomputation.

DRAM access of an operand without recomputation:

Scenario 1: When the blocker of an operand lies within its own dimensions, the operand's DRAM access is calculated as the product of: (1) its buffer size requirement, (2) the loop boundary of its blocker, and (3) the loop boundaries of all effective dimensions above the blocker. For operand A in Fig. 10, the buffer size requirement is $BS_A = i_G k_G k_D$, the loop boundary of its blocker is i_D, and there are no additional dimensions above i_2. Thus, the DRAM access for A is:

$$DA_A = BS_A i_D = i_G k_G k_D i_D \tag{5}$$

This corresponds to the total size of matrix A, indicating that each element is fetched from DRAM exactly once.

Scenario 2: When an operand is associated with the consumer operator and its blocker is a reduction dimension of the producer operator, the corresponding DRAM access is estimated as the product of: (1) its buffer size requirement, and (2) the loop boundaries of all effective dimensions above the blocker. Unlike Scenario 1, the loop boundary of the blocker is excluded, as the blocker is not directly related to the operand. For operand D in Fig. 10, where the blocker is k_2, the DRAM access is estimated as:

$$DA_D = BS_D l_D j_D i_D = l_G j_G l_D j_D i_D \tag{6}$$

This means the DRAM access equals i_D copies of matrix D (size $l_G j_G l_D j_D$), or each element is accessed i_D times.

DRAM access of an operand with recomputation: When producer operands A and B are not retained in the buffer, recomputing the intermediate result C requires reloading the corresponding tiles of A and B from DRAM. This recomputation overhead is already captured in the two scenarios above through effective dimensions. These dimensions change dynamically depending on whether recomputation is applied.

Overall DRAM access:

$$DA = DA_A + DA_B + DA_D + DA_E \tag{7}$$

5.4 Energy Consumption and Latency Estimation

Our energy consumption estimation builds upon existing models [50, 66, 73], augmented by our analytical DRAM access models. It includes three components: 1. DRAM - buffer traffic energy, depending on DA; 2. buffer - register file traffic energy, influenced by the stationary mode (e.g., weight stationary); and 3. computation energy, proportional to total MAC (Multiply-Accumulate) operations. The total MAC count is given by $N_{op1} + N_{op2}$ without recomputation, or $j_D N_{op1} + N_{op2}$ with recomputation, where $N_{op1} = ikl$ is the MAC count for operator 1 and $N_{op2} = ijl$ for operator 2. We evaluate all combinations of three popular stationary modes (weight, input, and output stationary) for both operators under each computation ordering and buffer management solution.

Similarly, latency estimation is based on established frameworks [9, 29, 73], and assumes the data load, execution, and data store are fully pipelined with double buffer. It is estimated as the maximum of DRAM access latency, proportional to DA under a given DRAM bandwidth, and computation latency, determined by the the PE array size and its utilization.

The energy and latency of softmax are captured in our performance model. We introduce a factor $c_{softmax}$ to account for the energy of softmax operations. This user-defined parameter depends on hardware implementation and accuracy requirements [43, 65].

Since only Op1 ($Q \times K^T$) requires softmax, its energy is modeled as $c_{softmax} il$ ($c_{softmax} il j_D$ for recomputation). We use $c_{\text{softmax}} = 10$, under which softmax energy is approximately one order of magnitude lower than PE-compute energy. For latency, we adopt a tile-level pipeline: the current tile proceeds to SFU immediately after Op1, while the logits matrix (L) from the previous tile is multiplied with V to perform $L \times V$. This pipeline enables latency hiding between consecutive tiles, consistent with FlashAttention [55]. To support this, sufficient SFU resources are allocated [33].

5.5 Accounting for Loop Order and Buffering Levels

The models described so far are in terms of tile sizes (loop boundaries) for a specific setting of loop order and buffering level. To capture the impact from different loop orders and buffering levels, we adopt the "ln – exp" trick presented in [28, 41]. Consider a performance metric such as buffer size requirement for operand A, $BS_A = k_D i_G k_G$, for the loop order and buffering level shown in Fig. 10. It can be rewritten in vector form as:

$$BS_A = \exp(q \cdot \ln(b)) \tag{8}$$

where the query vector and boundary vector are defined as:

$$q = [0, 1, 0, 0, 1, 1, 0, 0] \tag{9}$$

$$b = [i_D, k_D, l_D, j_D, i_G, k_G, l_G, j_G]^T \tag{10}$$

Likewise, buffer size requirements and DRAM accesses for any loop orders and buffering levels can be expressed in this form, where the **boundary vector** encodes the tiling decisions, while the **query vector** encodes loop ordering and buffering level decisions. Based on analytical formulas like Equation (8) and the process described in Section 5.4, latency and energy can be estimated for any legitimate loop orders, buffering levels and tiling without the need for "if-else" parsing [71] or tree traversal [73]. Please note that the model of [28, 41] is restricted to intra-operator dataflows while ours works for cross-operator fusion dataflows due to our significant extensions described in Sections 5.2 and 5.3.

6 Dataflow Optimization

6.1 Optimization by MMEE

An overview of our MMEE (Matrix Multiplication Encoded Enumeration) method is shown in Fig. 11. A key idea in MMEE is partitioning the decision space into two parts: (1) *loop orders and buffering levels*, whose candidate options are enumerated offline; and (2) *tiling configurations*, whose options are enumerated online. The rationale for offline enumeration is that most attention mechanisms share the same pseudo-nested-loop structure across workloads. Therefore, loop-order and buffering-level options can be enumerated once and reused. This offline phase takes 1.7 hours on an AMD Ryzen 7 7840H CPU and is a one-time cost. In contrast, tile sizes depend on workload-specific dimensions and must be enumerated online.

This decision space partitioning coincides with the "ln – exp"-based model by Equation (8), where a solution is evaluated by the inner product between the query vector, which encodes loop order and buffering level options, and the boundary vector, which indicates tiling configurations. Then, multiple loop ordering and

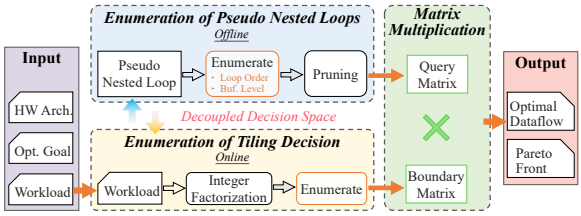


Figure 11: Overview of the MMEE dataflow optimization framework. The decision space is decoupled into two independently enumerated subspaces, represented in matrix form. Optimal dataflow solutions and performance tradeoffs are then obtained through matrix multiplication.

buffering level solutions can be described by a **query matrix** $Q = [q_1, q_2, ..., q_m]^T$, where each row is a query vector. Likewise, multiple tiling options can be represented by a **boundary matrix** $B = [b_1, b_2, ..., b_n]$, where each column corresponds a boundary vector. Then, solutions for all loop order, buffering level and tiling options can be evaluated by the product between Q and B as

$$\exp\left(\begin{bmatrix} q_1 \\ q_2 \\ \dots \\ q_m \end{bmatrix} \ln\left(\begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}\right)\right) = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ \dots & \dots & \dots & \dots \\ r_{m1} & r_{m2} & \dots & r_{mn} \end{bmatrix} \tag{11}$$

where r_{ij} is the result, such as buffer size requirement or DRAM access, of loop order and buffering level corresponding to q_i and tiling configuration b_j.

MMEE utilizes six types of query matrices: $Q_{BS,P}$ for producer buffer size requirements, $Q_{BS,C}$ for consumer buffer size requirements, Q_{DA} for DRAM access, $Q_{C,P}$ for producer computation latency, $Q_{C,C}$ for consumer computation latency, and Q_{BR} for traffic volume between the buffer and register files. In addition, a Q matrix encodes the option of whether recomputation is performed.

By integrating these query matrices with the boundary matrix B via Equation (11), and by applying the analytical models and pruning, energy and latency estimates can be computed for all enumerated solutions. This exhaustive enumeration enables the identification of optimal configurations for a variety of objectives.

Although matrix encoding also appears in [41], there are three key differences. (1) [41] targets only intra-operator dataflows, while MMEE supports cross-operator dataflows. In particular, the six query matrices in MMEE are redesigned to correctly handle fused operators. (2) [41] relaxes the problem into a continuous domain and requires a discretization step to produce integer solutions, making the optimization inherently heuristic. MMEE instead exhaustively enumerates and guarantees optimality. (3) MMEE incorporates a pruning technique, absent in [41], which accelerates enumeration-based search without compromising optimality.

6.2 Offline Symbolic Pruning

Some computation ordering and buffer management solutions can be pruned without affecting optimality. We propose a symbolic pruning technique that is independent of workloads and tiling solutions. Thus, pruning can be performed offline.

We first partition these solutions into groups based on whether recomputation is enabled (2 options) and combinations of intra-operator stationary options. Each operator has three stationary options (WS, OS and IS); two operators yield 9 combinations, resulting in 18 groups in total. Pruning is then carried out within each group individually.

For each group, the pruning is based on two criteria: buffer size and DRAM access. Even without specific tiling solutions, we can derive symbolic expressions for buffer size requirement and DRAM access per query matrix row and perform pairwise comparisons. For example, given two buffer management solutions s_u and s_v, if their buffer sizes are estimated as $BS_u = i_G k_G$ and $BS_v = i_G k_G i_D$, then $BS_u < BS_v$ holds for all valid tiling solutions. For any solution pair (s_u, s_v), s_v is inferior to s_u and pruned if

$$BS_v \geq BS_u \text{ and } DA_v > DA_u,$$
$$\text{or } BS_v > BS_u \text{ and } DA_v \geq DA_u \tag{12}$$

Applying the pruning reduces each query matrix per group from 20K rows to 58, revealing the large redundancy in the computation-ordering and buffer-management subspace.

6.3 Optimality of MMEE

MMEE guarantees to find the optimal latency-energy tradeoff for fused attention dataflow mapping according to the models described in Section 5. We will show that if a solution s_v is inferior according inequalities (12) then it must lead to an inferior solution in terms of energy and latency.

First, consider the energy part. Two solutions in the same group cost the same PE computation energy (as both have the same choice of recomputation) and the same buffer-register file traffic energy as they share the same intra-operator stationary option. Then, energy difference between them is determined by the DRAM access energy and DRAM-buffer energy. Since s_v must be worse in both DRAM access and buffer size, s_v must be inferior in the overall energy, considering that DRAM-buffer energy is proportional to buffer size.

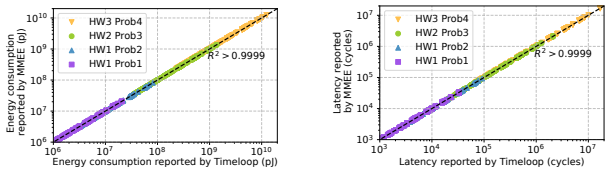
Second, consider latency, which is the maximum between computation latency and DRAM access latency. Solutions in the same group have identical computation latency. So the difference is decided by DRAM latency, i.e., DRAM access volume divided by bandwidth. Since s_v has higher DRAM access, then it must have inferior latency in the same group.

Overall, a solution pruned according to inequalities (12) is inferior in terms of both energy and latency in its group. Since all solutions are enumerated and only the energy-latency inferior solutions are pruned out, the pruning would retain all optimal solutions in terms of energy-latency tradeoff.

7 Evaluation

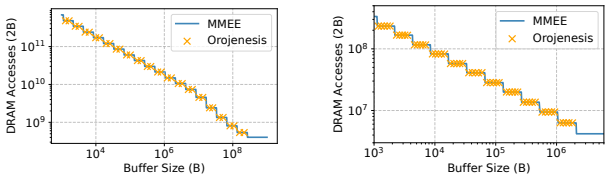
7.1 Experiment Setup

To evaluate MMEE across different hardware settings, we use two accelerator configurations. Accel. 1 follows the NVDLA design [49, 73], featuring 4 PE arrays, a 1 MB on-chip buffer, 60 GB/s DRAM bandwidth, and 32×32 PEs per array. Accel. 2 adopts a TPU-like design [30, 54], with 4 PE arrays, a 4 MB buffer, 128 GB/s DRAM bandwidth, and 128×128 PEs per array. Both accelerators operate at 1 GHz. Energy parameters (SRAM access and PE compute) are taken from [68] under 28 nm. All experiments are conducted on an AMD Ryzen 7 7840H CPU @ 3.8GHz.



(a) Energy consumption. (b) Latency.

Figure 12: Our model vs. Timeloop.



(a) Fusing 32k_4k_16k and 32k_16k_4k GEMMs. (b) Fusing 1k_1k_1k and 1k_1k_1k GEMMs.

Figure 13: DRAM access and buffer size requirement estimation between our model and NVIDIA's Orojenesis.

7.2 Model Validation

We validate our models for latency, energy, buffer size, and DRAM access using Timeloop [50], a widely adopted reference. A total of 1410 diverse intra-operator mappings are evaluated across three hardware configurations (HW1–HW3) and four matrix multiplication workloads (Prob1–Prob4), covering a broad performance range. Fig. 12 shows the results. Our models match Timeloop closely, achieving $R^2 > 0.9999$ for both latency and energy. Mean errors are near zero, with **the maximum errors** of only 0.5% in energy and 0.05% in latency.

We also compare our models with NVIDIA's Orojenesis [29], which estimates DRAM access and buffer size requirement for fusion dataflow mapping. Fig. 13 shows the comparison results for two workloads. The results show that mean errors are 0.33% and 0.25%, while **maximum errors** are 0.78% and 0.68% for the two workloads, respectively.

7.3 Dataflow Optimization for DRAM Access

Baselines. Orojenesis [29] explores minimal achievable DRAM access of attention fusion under various buffer sizes and is the state-of-the-art method. To clarify MMEE's sources of improvement, **we extend Orojenesis with two variants**: "O+BM" (Orojenesis enhanced with buffer management) and "O+BM+Re" (Orojenesis further enhanced with recomputation).

Workloads. Following Orojenesis, we evaluate attention computation of GPT-3-6.7B.

DRAM access. Fig. 14 shows DRAM access of MMEE across buffer sizes from 64 KB to 4 MB [26, 49, 75], achieving up to 1.30× reduction. "O+BM" and "O+BM+Re" provide sources of improvement. Buffer management reduces DRAM traffic by 1.14× at 64 KB, while recomputation achieves 1.20-1.31× reduction at larger buffer sizes.

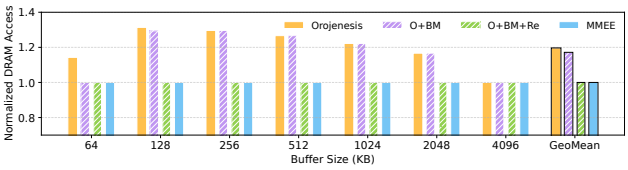


Figure 14: Fusing attention computation of GPT-3-6.7B.

Table 1: Dimensions across models and sequence lengths.

Dim.	BERT-Base			GPT-3-13B			PaLM-62B		
	512	4K	16K	2K	4K	16K	2K	4K	16K
I	512	4096	16384	2048	4096	16384	2048	4096	16384
K	64	64	64	128	128	128	256	256	256
L	512	4096	16384	2048	4096	16384	2048	4096	16384
J	64	64	64	128	128	128	256	256	256

At 4 MB, the buffer can hold all attention matrices, yielding no difference among the four mappers.

Runtime. For the fusion task in Fig. 14, Orojenesis evaluates 1.6×10^6 mappings in around 272 seconds. In contrast, MMEE evaluates 1.7×10^8 mappings within 0.96 seconds, making it **283× faster** than Orojenesis. This speedup comes from: (1) decoupling the decision space and precomputing all combinations of computation ordering and buffer management; (2) matrix-based parallel evaluation; (3) pruning that further reduces evaluation cost.

7.4 Dataflow Results on Energy and Latency

Baselines. We compare our approach against three state-of-the-art cross-operator dataflow mappers FLAT [33], Chimera [74] and TileFlow [73]. Specifically, we select R-Gran from FLAT [33], which offers the most fine-grained dataflow suited for limited on-chip buffer resources. Chimera [74] is not open-sourced and is reproduced using TileFlow's implementation [73]. Orojenesis is excluded as it does not report energy or latency results.

For a fair comparison with TileFlow, we use its official GitHub implementation and evaluate both TileFlow and MMEE on Accel. 1 and Accel. 2 under identical hardware settings (Section 7.1). TileFlow must be configured as either energy- or latency-driven before search, whereas MMEE evaluates all dataflows and metrics simultaneously. For consistency, we group MMEE results into the same two modes. TileFlow's reported runtime only accounts for MCTS-based tiling search, and buffer management and computation ordering decisions are pre-searched by a genetic algorithm and fixed in the released code. We use TileFlow's default timeout configuration and ensure convergence.

Workloads. Attention computations from three models of varying scales are used to evaluate the effectiveness of the proposed methods. From small to large, the selected models are BERT-Base [21], GPT-3-13B [8] and PaLM-62B [16]. In addition to each model's standard sequence length, longer sequences are also explored to assess the scalability of the evaluated methods. The corresponding attention dimensions are listed in Table 1.

Energy and latency. Figs. 15 and 16 present results for energy consumption and latency on Accel. 1 and Accel. 2. Results for MMEE and FLAT from our model, while others use TileFlow's model. To clarify the sources of energy improvement and differentiate between compute-bound and memory-bound scenarios, the corresponding energy and latency breakdowns are provided. Absolute values are reported in Table 2.

Table 2: Absolute energy consumption and latency values for MMEE in Figs. 15 and 16. Each mapping solution is represented in energy/latency (mJ/ms). Results for other methods can be derived using the relative ratios in the figures.

Model	Seq Len	Accel. 1 (mJ/ms)		Accel. 2 (mJ/ms)	
		E-Driven	L-Driven	E-Driven	L-Driven
BERT -Base	512	1.11/0.10	1.11/0.10	0.92/0.03	0.92/0.03
	4K	57.14/6.29	57.14/6.29	38.34/0.79	45.33/0.54
	16K	890.60/100.66	890.60/100.66	588.49/12.58	700.33/6.88
GPT -3-13B	2K	129.75/12.23	129.75/12.23	65.50/1.80	65.50/1.80
	4K	505.86/46.84	505.86/46.84	248.14/6.23	248.14/6.23
	16K	7936.0/724.2	7936.0/724.2	3804.0/87.8	3804.0/87.8
PaLM -62B	2K	268.68/27.96	268.68/27.96	132.03/4.98	180.58/3.93
	4K	1060.8/109.6	1060.8/109.6	505.9/18.3	700.1/14.2
	16K	16807/1727	16807/1727	7829/275	10936/208

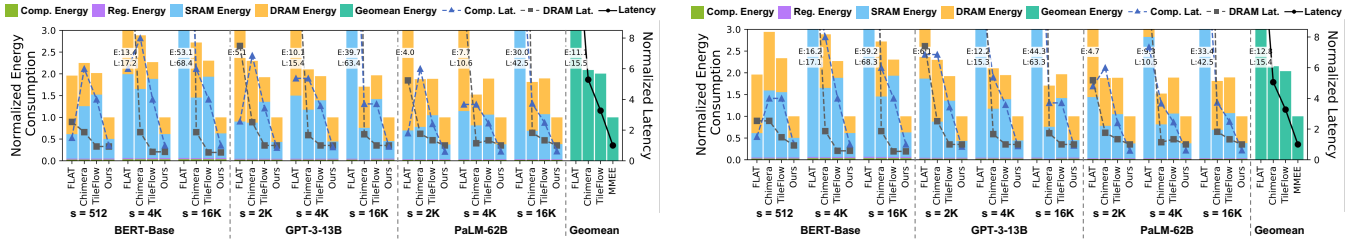
Accel. 1. When optimizing for either energy consumption or latency, MMEE achieves average reductions of 50% in energy consumption and 69% in latency, compared to TileFlow.

Accel. 2. In energy-driven mode, MMEE achieves an average 48% energy reduction and an average 31% latency reduction, relative to TileFlow. When optimizing latency, MMEE shows 40% less energy and 40% less latency on average.

The energy breakdowns in Figs. 15 and 16 highlight the sources of MMEE's energy savings. Among the dominant components, SRAM and DRAM energy, MMEE consistently achieves equal or lower consumption compared to other methods. For example, while TileFlow occasionally matches MMEE's DRAM energy (e.g., BERT-Base-512 on Accel. 1), it incurs significantly higher SRAM energy due to its reliance on a fixed and less efficient template, derived from genetic algorithms, for managing data movement between SRAM and register files. Moreover, for PaLM-62B-16K, TileFlow consumes more DRAM energy, stemming from its limited buffer management exploration and lack of recomputation.

The latency improvements of MMEE stem from two key factors: higher compute utilization and reduced DRAM access. Compute utilization [33] measures how closely a dataflow approaches the arithmetic optimum. As shown in Fig. 17, MMEE achieves significantly higher compute utilization than TileFlow. TileFlow's low utilization arises from its MCTS-based tiling search, which under-utilizes PE arrays, particularly on Accel. 1. This under-utilization leads to only 25% compute utilization and higher latency in compute-bound cases. In memory-bound scenarios, e.g., PaLM-62B models under latency-driven optimization on Accel. 2, MMEE reduces DRAM access via recomputation, which directly lowers overall latency.

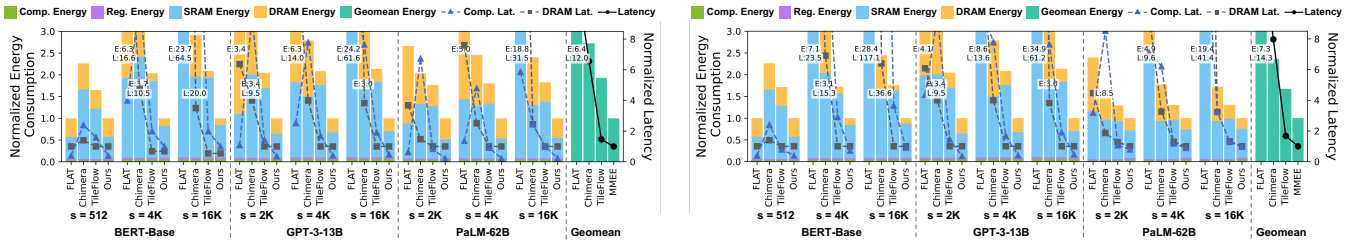
Runtime. We compare the runtime of TileFlow and MMEE across all experiments in Figs. 15 and 16. TileFlow is configured separately for latency- and energy-driven modes, and the reported runtime aggregates results across three models. Runtime results for Chimera and FLAT are not available, as neither their code is open-sourced nor is their runtime reported. Overall, our approach achieves significant runtime improvements over TileFlow, providing speedups of **64×** and **287×** on Accel. 1 and Accel. 2, respectively.



(a) Energy-driven optimization (E-Driven).

(b) Latency-driven optimization (L-Driven).

Figure 15: Dataflow optimization results on energy and latency for Accel. 1. The overall energy is the sum of four components, and the overall latency is the maximum of computation latency (Comp. Lat.) and DRAM latency (DRAM Lat.).



(a) Energy-driven optimization (E-Driven).

(b) Latency-driven optimization (L-Driven).

Figure 16: Dataflow optimization results on energy and latency for Accel. 2.

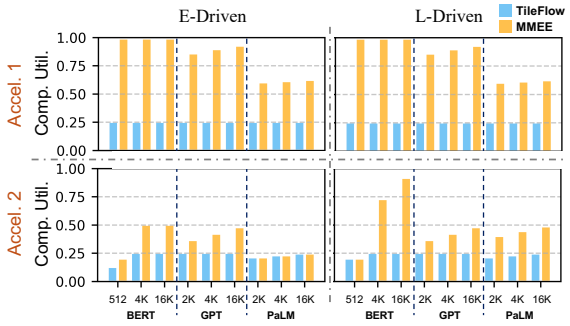
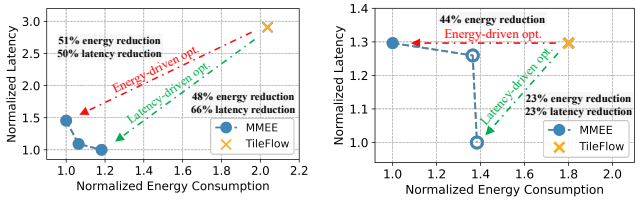


Figure 17: Compute utilization of TileFlow and MMEE.



(a) BERT-Base-4096.

(b) PaLM-62B-4096.

Figure 18: Energy-latency trade-off of the attention fusion.

7.5 Energy-Latency Trade-off

MMEE supports energy or latency optimization in isolation and also generates the complete energy–latency Pareto frontier.

Baseline and workload. TileFlow is configured with energy-driven and latency-driven optimizations, providing baseline comparisons for evaluating energy-latency trade-offs. We evaluate the fusion of attention operations in BERT-Base and PaLM-62B models on Accel. 2, each with a sequence length of 4096.

Trade-off results. Fig. 18 shows the Pareto front (blue dashed lines) produced by MMEE. Red and green arrows denote energy- and latency-driven solutions, respectively. Blue filled and unfilled circles represent recomputation-disabled and -enabled solutions, respectively. We make three key observations: (1) The Pareto front is sparse. (2) Points along the Pareto curve are highly sensitive, meaning optimizing energy solely increases latency, and vice versa. (3) Recomputation reduces latency and expands the Pareto frontier.

7.6 Evaluation on GPU

To assess the efficiency of dataflow mappers on real hardware, we deploy the dataflows generated by MMEE and TileFlow on an NVIDIA A100-40GB GPU and use Triton to compile them into CUDA kernels [62]. We use the official Triton implementation of the FlashAttention-2 algorithm as the Triton FA2 baseline [18, 51]. FA2 is adopted as the primary baseline because it is a state-of-the-art attention implementation for the Ampere architecture and is widely used in serving systems such as vLLM [39] and SGLang [72]. We further enable Triton autotuning and search 10 combinations of `num_warps` and `num_stages`, requiring 31 seconds on average. Table 3 summarizes the results. Overall, MMEE achieves a 2.56× speedup over TileFlow and a 1.18× speedup over Triton FA2, enabled by more flexible buffer management and tiling choices. GPT workloads are particularly sensitive to autotuning.

Table 3: Performance (ms) of MMEE on an A100 GPU.

Model	BERT-Base			GPT-3-13B			PaLM-62B		
Seq. Len.	512	4K	16K	2K	4K	16K	2K	4K	16K
TileFlow	0.05	0.64	2.45	0.55	1.14	4.24	1.63	3.23	12.83
Triton FA2	0.03	0.19	0.75	0.40	0.73	2.43	OOM	OOM	OOM
Autotune	0.03	0.19	0.74	0.31	0.61	2.13	OOM	OOM	OOM
MMEE	0.03	0.18	0.69	0.27	0.60	2.05	0.45	0.89	3.54

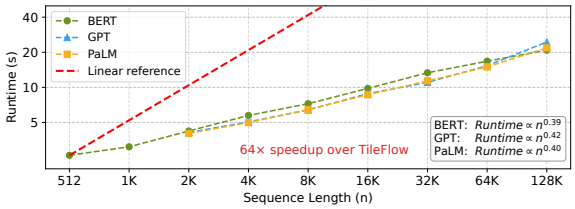


Figure 19: Runtime of MMEE vs. sequence length (log-log).

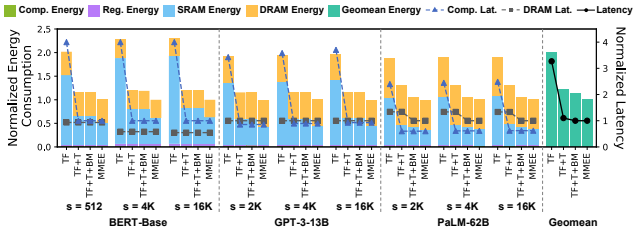


Figure 20: Analysis of decision elements.

7.7 Runtime Scalability

Fig. 19 shows MMEE's runtime and scaling with sequence length on Accel. 1. MMEE scales sub-linearly; at 128K, runtime remains below 25 seconds. Power-law fitting further quantifies this scalability. Due to decision-space decoupling and an efficient performance model, online runtime is dominated by tiling enumeration. For attention workloads, factorization complexity ($\propto n^{0.4}$ on average) grows much slower than quadratic compute complexity.

7.8 Sensitivity Analysis

7.8.1 Sensitivity to key decision elements. Fig. 20 quantifies the impact of three key decision elements on dataflow quality. "TF", "TF+T", and "TF+T+BM" denote TileFlow, TileFlow enhanced with tiling enumeration, and TileFlow enhanced with both tiling and buffer management enumeration, respectively. Fusion is under the energy-driven optimization on Accel. 1. TileFlow primarily suffers from an inefficient tiling search. Consequently, "TF+T" achieves 39% lower energy and 66% lower latency than TileFlow. Building upon this, flexible buffer management further reduces energy and latency by 7% and 9%, respectively. Enabling computation-ordering exploration yields an additional 11% energy reduction.

7.8.2 Sensitivity to pruning. In Section 6.2, pruning accelerates dataflow exploration. To verify its effectiveness, we repeat all experiments in Figs. 15 and 16 without pruning. The results show that pruning does not compromise optimality, while achieving significant speedups of 347× on Accel. 1 and 221× on Accel. 2, compared to MMEE without pruning.

Table 4: MMEE results across different hardware designs.

HW Design	PE Array Size	#PE Array	Buffer Size	DRAM Bandwidth	TileFlow (E/L)	MMEE (E/L)
Coral [27]	16×16	1	32 KB	1.6 GB/s	1.95/1.59	1/1
Design [75]	32×32	1	512 KB	2 GB/s	2.24/1.18	1/1
SET [9, 26]	32×32	16	16 MB	8 GB/s	4.17/2.56	1/1

Table 5: Results on the high-bandwidth configuration, normalized to MMEE (1.00/1.00 for energy/latency).

Model	Seq. Len.	TileFlow E-Driven	TileFlow L-Driven
BERT-Base	512	1.70/1.00	1.92/1.17
	4K	1.86/1.00	1.30/1.30
	16K	1.89/1.00	1.00/1.32
GPT-3-13B	2K	1.68/1.00	1.90/1.30
	4K	1.71/1.00	1.54/1.31
	16K	1.72/1.00	1.55/1.33
PaLM-62B	2K	1.70/1.34	2.67/1.34
	4K	1.71/1.34	1.71/1.34
	16K	1.71/1.33	1.71/1.33

Table 6: Optimization for convolution chains and two GEMMs.

Workload		Problem Shape	Baseline (E/L)	MMEE (E/L)
Conv. Chain	CC1 [73]	$[112^2,64,192,128,3^2,1^2]$	2.34/1.16	1/1
	CC2 [73]	$[56^2,64,64,64,1^2,1^2]$	1.20/1.50	1/1
Two GEMMs	MLP [74]	[768,64,384,64]	1.93/1.00	1/1
	FFN [21]	[2048,768,3072,768]	1.08/1.14	1/1

7.8.3 Sensitivity to hardware design. To demonstrate the generality of MMEE, we evaluate it on three hardware designs from both industry and academia [9, 26, 27, 75], as listed in Table 4. Entries for TileFlow and MMEE report normalized energy and latency values. Compared to TileFlow, MMEE achieves substantial reductions in energy and latency.

We further evaluate MMEE on an H100-like high-bandwidth configuration with 114 PE arrays, 32×32 PEs per array, a 256 KB buffer per array, 200 TFLOPS of compute, and 2 TB/s of memory bandwidth, as summarized in Table 5. Compared with TileFlow, MMEE achieves 1.74×/1.10× reductions in energy/latency under energy-driven optimization, and 1.65×/1.30× reductions under latency-driven optimization.

7.9 Applicability to Fusion Beyond Attention

MMEE supports fusion patterns representable as two dependent GEMMs. Unlike attention, generic two-GEMM fusion does not require the additional No-Psum-Propagation constraint. We further evaluate MMEE on two convolutions and two GEMMs in Table 6 using Accel. 1. For convolution chains, convolutions are converted to GEMMs via im2col, with shapes defined by input height/width, input channels, output channels of both convolutions, and kernel sizes. For two GEMMs, shapes are $[I, K, L, J]$. The baseline is the better result between TileFlow and intra-operator optimization. Overall, MMEE consistently achieves the best energy–latency tradeoffs.

8 Discussion

Beyond its role as a standalone dataflow optimizer, MMEE can be served as a high-level optimization pass for a modern MLIR-based compiler stack [1–3, 40, 62]. In this workflow, MMEE sits between the high-level dialect, which provides the computational graph, and the low-level backend dialect, which handles code generation. By matrix-form enumeration, MMEE rapidly determines the optimal tiling, computation ordering and buffer management for a given hardware template. These parameters can be leveraged in generating highly efficient kernels and code. This integration allows MMEE to bridge the gap between abstract operator representations and the compiler, navigating complex cross-operator spaces that standard compilers fall short of.

9 Related Work

Intra-operator mapping, which optimizes individual operators, has been widely studied [14, 23, 24, 32, 36–38, 53, 63, 67]. Prior work includes heuristic [12, 15, 23, 30] and automated approaches [28, 41, 44, 60]. For cross-operator dataflow mapping, early efforts [5, 59] demonstrated CNN fusion. Recent frameworks [25, 31, 75, 76] extend fusion strategies to general DNNs. Orojenesis [29] targets matrix multiplication fusion via exhaustive tiling guided by computation ordering templates. TileFlow [73] introduces a tree representation and uses genetic algorithms and Monte Carlo Tree Search to explore tiling, computation ordering, and buffer management. SET [9] creates a tree notation emphasizing batch splitting, while Crane [26] improves scheduling quality and speed by hierarchical table formats. FFM [6] explores inter-operator space but remains limited in intra-operator aspects such as stationary modes. FlashAttention [18, 19] applies the online softmax [46] to reduce IO access. FLAT [33] exhaustively searches tiling space. FuseMax [48] leverages an extended Einsum notation to jointly optimize PE array design and attention dataflow.

10 Conclusion

This work proposes **MMEE**, a framework for efficiently exploring attention fusion dataflows. MMEE systematically captures key design elements and introduces a traversal-free, branch-free analytical performance model. By combining pruning with matrix-based evaluation, MMEE rapidly enumerates various design tradeoffs. Experimental results show that MMEE outperforms state-of-the-art methods in both solution quality and optimization speed.

Acknowledgments

This work is partially supported by the NSF (CCF-2106725, CCF-2212346, CCF-2425399, CCF-2529764, CCF-2529763) and the Hans Fischer Senior Fellowship of the Technical University of Munich.

References

[1] 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

[2] Dennis Abts, Garrin Kimmell, Andrew Ling, John Kim, Matt Boyd, Andrew Bitar, Sahil Parmar, Ibrahim Ahmed, Roberto DiCecco, David Han, et al. 2022. A Software-defined Tensor Streaming Multiprocessor for Large-scale Machine Learning. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*. 567–580.

[3] Advanced Micro Devices, Inc. 2023. MLIR-AIE: MLIR-based Toolchain for AI Engine-enabled Devices. https://xilinx.github.io/mlir-aie/. Accessed: 2024-05-22.

[4] Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-query Transformer Models from Multi-head Checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 4895–4901.

[5] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN Accelerators. In *International Symposium on Microarchitecture*.

[6] Tanner Andrulis, Michael Gilbert, Vivienne Sze, and Joel S Emer. 2026. Fast and Fusiest: An Optimal Fusion-Aware Mapper for Accelerator Modeling and Evaluation. *arXiv preprint arXiv:2602.15166* (2026).

[7] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The Long-document Transformer. *arXiv preprint arXiv:2004.05150* (2020).

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-shot Learners. *Advances in Neural Information Processing Systems* (2020).

[9] Jingwei Cai, Yuchen Wei, Zuotong Wu, Sen Peng, and Kaisheng Ma. 2023. Inter-layer Scheduling Space Definition and Exploration for Tiled Accelerators. In *International Symposium on Computer Architecture*.

[10] Xuyi Cai, Ying Wang, and Lei Zhang. 2022. Optimus: An Operator Fusion Framework for Deep Neural Networks. *Transactions on Embedded Computing Systems* (2022).

[11] Chia-Wei Chang, Jing-Jia Liou, Chih-Tsun Huang, Wei-Chung Hsu, and Juin-Ming Lu. 2023. MultiFuse: Efficient Cross Layer Fusion for DNN Accelerators With Multi-Level Memory Hierarchy. In *International Conference on Computer Design*.

[12] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning. *SIGARCH Computer Architecture News* (2014).

[13] Xiaoming Chen, Yinhe Han, and Yu Wang. 2020. Communication Lower Bound in Convolution Accelerators. In *International Symposium on High Performance Computer Architecture*.

[14] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2016. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *Journal of Solid-State Circuits* (2016).

[15] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *Journal on Emerging and Selected Topics in Circuits and Systems* (2019).

[16] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research* (2023).

[17] Alexis Conneau and Guillaume Lample. 2019. Cross-Lingual Language Model Pretraining. *Advances in Neural Information Processing Systems* (2019).

[18] Tri Dao. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *arXiv preprint arXiv:2307.08691* (2023).

[19] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-efficient Exact Attention with IO-Awareness. *Advances in Neural Information Processing Systems* (2022).

[20] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. 2021. Compute and Energy Consumption Trends in Deep Learning Inference. *arXiv preprint arXiv:2109.05472* (2021).

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

[22] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An Image Is Worth 16x16 Words: Transformers

for Image Recognition at Scale. *arXiv preprint arXiv:2010.11929* (2020).

[23] Clément Farabet, Berin Martini, Benoit Corda, Polina Akselrod, Eugenio Culurciello, and Yann LeCun. 2011. NeuFlow: A Runtime Reconfigurable Dataflow Processor for Vision. In *Computer Vision and Pattern Recognition Workshops*.

[24] Yonggan Fu, Yongan Zhang, Yang Zhang, David Cox, and Yingyan Lin. 2021. Auto-NBA: Efficient and Effective Search Over the Joint Space of Networks, Bitwidths, and Accelerators. In *International Conference on Machine Learning*.

[25] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. TANGRAM: Optimized Coarse-grained Dataflow for Scalable NN Accelerators. In *International Conference on Architectural Support for Programming Languages and Operating Systems*.

[26] Yu Gong, Lingyi Huang, Haodong Chang, Rongjian Liang, Cheng Yang, Zhexiang Tang, Jiang Hu, and Bo Yuan. 2025. Crane: Inter-Layer Scheduling Framework for DNN Inference and Training Co-Support on Tiled Architecture. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*. 1250–1263.

[27] Google. 2025. Coral NPU. https://developers.google.com/coral/guides/intro. Accessed: 2025-10-16.

[28] Qijing Huang, Minwoo Kang, Grace Dinh, Thomas Norell, Aravind Kalaiah, James Demmel, John Wawrzynek, and Yakun Sophia Shao. 2021. Cosa: Scheduling by Constrained Optimization for Spatial Accelerators. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 554–566.

[29] Qijing Huang, Po-An Tsai, Joel S Emer, and Angshuman Parashar. 2024. Mind the Gap: Attainable Data Movement and Operational Intensity Bounds for Tensor Algorithms. In *International Symposium on Computer Architecture*.

[30] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *International Symposium on Computer Architecture*.

[31] Sheng-Chun Kao, Xiaoyu Huang, and Tushar Krishna. 2022. DNNFuser: Generative Pre-trained Transformer as a Generalized Mapper for Layer Fusion in DNN Accelerators. *arXiv preprint arXiv:2201.11218* (2022).

[32] Sheng-Chun Kao and Tushar Krishna. 2020. GAMMA: Automating the Hw Mapping of DNN Models on Accelerators via Genetic Algorithm. In *International Conference on Computer-Aided Design*.

[33] Sheng-Chun Kao, Suvinay Subramanian, Gaurav Agrawal, Amir Yazdanbakhsh, and Tushar Krishna. 2023. FLAT: An Optimized Dataflow for Mitigating Attention Bottlenecks. In *International Conference on Architectural Support for Programming Languages and Operating Systems*.

[34] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. 2023. On the Computational Complexity of Self-Attention. In *International Conference on Algorithmic Learning Theory*.

[35] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The Efficient Transformer. *arXiv preprint arXiv:2001.04451* (2020).

[36] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*. ACM, 754–768.

[37] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. 2020. MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. *IEEE Micro* 40, 3 (2020), 20–29.

[38] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping Over DNN Accelerators via Reconfigurable Interconnects. *International Conference on Architectural Support for Programming Languages and Operating Systems* (2018).

[39] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th symposium on operating systems principles*. 611–626.

[40] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2–14.

[41] Rongjian Liang, Jianfeng Song, Yuan Bo, and Jiang Hu. 2022. Deep Learning Toolkit-accelerated Analytical Co-optimization of CNN Hardware and Dataflow. In *International Conference on Computer-Aided Design*.

[42] Hezheng Lin, Xing Cheng, Xiangyu Wu, and Dong Shen. 2022. CAT: Cross Attention in Vision Transformer. In *International Conference on Multimedia and Expo*.

[43] Jiawei Lin, Yuanlong Li, Guokai Chen, and Thomas Bourgeat. 2025. SystolicAttention: Fusing FlashAttention within a Single Systolic Array. *arXiv preprint arXiv:2507.11331* (2025).

[44] Liqiang Lu, Naiqing Guan, Yuyue Wang, Liancheng Jia, Zizhang Luo, Jieming Yin, Jason Cong, and Yun Liang. 2021. Tenet: A Framework for Modeling Tensor Dataflow Based on Relation-centric Notation. In *International Symposium on Computer Architecture*.

[45] Linyan Mei, Koen Goetschalckx, Arne Symons, and Marian Verhelst. 2023. Defines: Enabling Fast Exploration of the Depth-First Scheduling Space for DNN Accelerators Through Analytical Modeling. In *International Symposium on High-Performance Computer Architecture*.

[46] Maxim Milakov and Natalia Gimelshein. 2018. Online Normalizer Calculation for Softmax. *arXiv preprint arXiv:1805.02867* (2018).

[47] S Naveen, MS S Ram Kiran, M Indupriya, TV Manikanta, and PV Sudeep. 2021. Transformer Models for Enhancing AttnGAN Based Text to Image Generation. *Image and Vision Computing* (2021).

[48] 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. In *International Symposium on Microarchitecture*.

[49] NVIDIA. 2017. NVDLA Deep Learning Accelerator. http://nvdla.org. Accessed: 2025-06-16.

[50] 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 *International Symposium on Performance Analysis of Systems and Software*.

[51] Philippe Tillet. 2020. Triton Getting Started Tutorials. https://triton-lang.org/main/getting-started/tutorials/. Accessed: 2025-07-27.

[52] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the Limits of Transfer Learning With a Unified Text-To-Text Transformer. *Journal of Machine Learning Research* (2020).

[53] Brandon Reagen, José Miguel Hernández-Lobato, Robert Adolf, Michael Gelbart, Paul Whatmough, Gu-Yeon Wei, and David Brooks. 2017. A Case for Efficient Accelerator Design Space Exploration via Bayesian Optimization. In *International Symposium on Low Power Electronics and Design*.

[54] Kiran Seshadri, Berkin Akin, James Laudon, Ravi Narayanaswami, and Amir Yazdanbakhsh. 2022. An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks. In *2022 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 79–91.

[55] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. 2024. FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision. *Advances in Neural Information Processing Systems* 37 (2024), 68658–68685.

[56] Haikuo Shao, Jinming Lu, Meiqi Wang, and Zhongfeng Wang. 2023. An Efficient Training Accelerator for Transformers with Hardware-algorithm Co-optimization. *Transactions on Very Large Scale Integration Systems* (2023).

[57] Noam Shazeer. 2019. Fast Transformer Decoding: One Write-head is All You Need. *arXiv preprint arXiv:1911.02150* (2019).

[58] Zhuoran Shen, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. 2021. Efficient Attention: Attention With Linear Complexities. In *Conference on Applications of Computer Vision*.

[59] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. 2017. PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning. In *International Symposium on High Performance Computer Architecture*.

[60] Arthur Stoutchinin, Francesco Conti, and Luca Benini. 2019. Optimally Scheduling CNN Convolutions for Efficient Memory Access. *arXiv preprint arXiv:1902.01492* (2019).

[61] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2020. Long Range Arena: A Benchmark for Efficient Transformers. *arXiv preprint arXiv:2011.04006* (2020).

[62] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. 2019. Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. 10–19.

[63] Rangharajan Venkatesan, Yakun Sophia Shao, Miaorong Wang, Jason Clemons, Steve Dai, Matthew Fojtik, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. 2019. MAGNet: A Modular Accelerator Generator for Neural Networks. In *International Conference on Computer-Aided Design*.

[64] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. Spatten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning. In *International Symposium on High-Performance Computer Architecture*.

[65] Zhigang Wei, Aman Arora, Pragenesh Patel, and Lizy John. 2020. Design Space Exploration for Softmax Implementations. In *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 45–52.

[66] Yannan Nellie Wu, Joel S Emer, and Vivienne Sze. 2019. Accelergy: An Architecture-level Energy Estimation Methodology for Accelerator Designs. In *International Conference on Computer-Aided Design*.

[67] Qingcheng Xiao, Size Zheng, Bingzhe Wu, Pengcheng Xu, Xuehai Qian, and Yun Liang. 2021. HASCO: Towards Agile Hardware and Software Co-design for Tensor Computation. In *International Symposium on Computer Architecture*.

[68] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, et al. 2020. Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators. In *International Conference on Architectural Support for Programming Languages and Operating Systems*.

[69] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. 2020. Gobo: Quantizing Attention-based NLP Models for Low Latency and Energy Efficient Inference. In *International Symposium on Microarchitecture*.

[70] Bowen Zhang, Shuyang Gu, Bo Zhang, Jianmin Bao, Dong Chen, Fang Wen, Yong Wang, and Baining Guo. 2022. StyleSwin: Transformer-Based GAN for High-Resolution Image Generation. In *Conference on Computer Vision and Pattern Recognition*.

[71] Yang Zhao, Chaojian Li, Yue Wang, Pengfei Xu, Yongan Zhang, and Yingyan Lin. 2020. DNN-Chip Predictor: An Analytical Performance Predictor for DNN Accelerators with Various Dataflows and Hardware Architectures. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1593–1597.

[72] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. 2024. SGLang: Efficient Execution of Structured Language Model Programs. *Advances in neural information processing systems* 37 (2024), 62557–62583.

[73] 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 *International Symposium on Microarchitecture*.

[74] 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 *International Symposium on High-Performance Computer Architecture*.

[75] Shixuan Zheng, Xianjue Zhang, Daoli Ou, Shibin Tang, Leibo Liu, Shaojun Wei, and Shouyi Yin. 2020. Efficient Scheduling of Irregular Network Structures on CNN Accelerators. *Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

[76] Xilang Zhou, Shuyang Li, Haodong Lu, and Kun Wang. 2024. PipeFuser: Building Flexible Pipeline Architecture for DNN Accelerators via Layer Fusion. In *Asia and South Pacific Design Automation Conference*.

[77] Zikang Zhou, Xuyang Duan, Kaiqi Chen, Yaqi Chen, and Jun Han. 2024. ML-Fusion: Determining Memory Levels for Data Reuse Between DNN Layers. In *Proceedings of the Great Lakes Symposium on VLSI*.