An Optimized Dataflow for Mitigating Attention Performance Bottlenecks
Sheng-Chun Kao¹, Suvinay Subramanian², Gaurav Agrawal² and Tushar Krishna¹
¹Georgia Institute of Technology
²Google
skao6@gatech.edu, tushar@ece.gatech.edu
²{suvinay, chipguy} @google.com

Abstract
Attention mechanisms form the backbone of state-of-the-art machine learning models for a variety of tasks. Deploying them on deep neural network (DNN) accelerators, however, is prohibitively challenging especially under long sequences, as this work identifies. This is due to operators in attention layers exhibiting limited reuse opportunities and quadratic growth in memory footprint, leading to severe memory-boundedness. To address this, we introduce a new attention-tailored dataflow, termed FLAT, which identifies fusion opportunities within the attention layer, and implements an on-chip memory-aware interleaved execution and tiling mechanism. FLAT increases the effective memory bandwidth by efficiently utilizing the high-bandwidth, low-capacity on-chip buffer and thus achieves better run time and compute resource utilization. In our evaluation, FLAT achieves 1.94x and 1.76x speedup and 49% and 42% of energy reduction comparing to baseline execution over state-of-the-art edge and cloud accelerators.

1. Introduction

Attention mechanisms, which form a key building block within Transformer models, have enabled state-of-the-art performance across a wide range of machine learning (ML) tasks – from natural language processing (NLP) [80], to object detection [14,74,97], image generation [16,31,60], and music synthesis [38,39] – and are expected to serve as the foundation for a whole new generation of machine learning models in the coming years. A key attribute of attention-based models is the sequence length, which is the length of input sentences, documents, or images. There is growing importance and surging interest in the ML community for long-sequence attention-based models. Examples include image generation (sequence length N=12K [22,25,27]), paragraph summarization (N=64K [50]), language modeling (N=69K [64]), and more upcoming new applications and benchmark suites [78].

Given the rising importance and growth of attention-based models [1,15], this work focuses on identifying key bottlenecks and opportunities for hardware acceleration (for inference), similar to how the community has been focusing on convolutions [19,20,54,70] and recommendation models [40]. Figure 1 shows the structure of attention-based models. The main kernel is called an attention layer, which comprises of six MatMul operations. At first glance, this breakdown suggests that accelerating attention layers by running the MatMul operators over state-of-the-art GEMM (General Matrix Multiply) accelerators (such as a TPU [44]) seems like a natural fit and is in fact the practice today. Unfortunately, as this work recognizes, the Logit (L) and Attend (A) MatMul operators exhibit different computational properties from those in traditional convolution (CONV) or fully-connected (FC) operators, and
are accordingly more challenging to accelerate (especially at long sequence lengths) limiting overall performance.

Operational intensity is the algorithmic maximum reuse [52] that can be extracted from an operator. Figure 2 plots the roofline performance for operators within Transformer models and contrasts it with CONV operators from ResNet50, all running on TPU-v3. CONV and FC (with batching) operators exhibit high operational intensity (at the roofline), indicating they are typically compute-bound. This is because they are activation-weight tensor operations, and allow weight reuse across multiple activations via batching and dataflow optimizations [2, 21, 52]. This reuse opportunity has indeed been the reason for the success of DNN accelerators in reducing expensive off-chip memory accesses for these operators. By contrast, the computation of the attention matrix within an attention layer involves L and A which are activation-activation tensor operations. Both these operators, unfortunately, exhibit low operational intensity, putting them in the memory-bandwidth (BW) bound regime (red stars in Figure 2). These operators contribute to high under-utilization leading to performance drop as our evaluations demonstrate. Moreover, the intermediate tensor between these two operators grows quadratically in size with the sequence length (more than 8GB for BERT [80], TrXL [27], and XLM [55] at 16K seq length as we quantify in §3.2), adding further pressure on the memory system as it exceeds the on-chip memory capacity of most datacenter accelerators, necessitating off-chip accesses.

To address these challenges, this work identifies a unique opportunity to enhance the effective operational intensity of L and A and extract on-chip reuse from the intermediate tensor via a fusion technique. While fusion itself is a known technique [58], it is limited in practice today in ML frameworks to fusing MatMul operators with element-wise operators (such as activation functions) [7, 11, 18, 51]. Fusion is not employed between MatMul operators as articulated in recent work [58]. The key reason is that (typical) MatMul operators already provide sufficient reuse benefits: the additional complexity of fusion (to maintain dependence and stage data) ends up negatively impacting register and cache usage [58]. Fusion opportunities have not been explored for attention-based models in literature, given how recent these models are. This is the first work to identify that attention MatMuls can actually benefit significantly from fusion, leading to additional, currently unexploited, reuse opportunities within attention layers.

To implement L and A fusion, we propose a new dataflow called **Fused Logit Attention Tiling** (FLAT). FLAT identifies that while fusion itself address the op-intensity challenge, it is important to tile and schedule the computations appropriately to (i) respect data dependencies and (ii) tackle the quadratic memory growth of the intermediate tensor. In particular, by appropriately sizing the fused-operator tile for any given target accelerator, we demonstrate that we are able to stage it completely within the limited on-chip buffer for both datacenter and edge configurations, eliminating all off-chip accesses for the intermediate tensor. This allows FLAT to easily scale to large sequence lengths without becoming BW bound.

We make the following contributions:

- We characterize the operational intensity of operators within attention layers, identifying the limitations posed by on-chip buffer sizes and memory bandwidth in running long sequence lengths (§3).
- We systematically explore fusion opportunities within the attention layer and make a case for fusing L and A (§4).
- We implement a new dataflow optimization approach FLAT that (i) preserves the softmax dependence within the fused operator and (ii) enables reuse of the quadratically growing tensor from low-capacity but high-bandwidth on-chip memory, eliminating the pressure on off-chip BW (§5).
- We introduce a map-space exploration framework for FLAT to find the optimal loop orders and tile sizes to optimize the performance metrics of interest (e.g., run time, energy) subject to varying hardware resource constraints (e.g., number of PEs, on-chip memory capacity) of the target accelerator for different models and usage scenarios (edge / cloud) (§6).
- FLAT achieves 1.75x and 1.65x speedup and 44% and 55% of energy reduction in edge and cloud accelerator configurations respectively, compared to an optimized baseline dataflow over a suite of Attention-based models [27, 55, 56, 65, 80] (§7).

2. Background

2.1. Attention Mechanism

The attention mechanism [12] has emerged as a fundamental building block of several ML models and is growing popular by the day. Attention-based models have demonstrated state-of-the-art performance across a wide range of tasks—from neural machine translation [63], to object detection [14, 74, 97], image generation [16, 31, 60], and music synthesis [38, 39], and many other applications [23]—often surpassing the performance of traditional CNN/RNN-based models.

2.1.1. Terminology for Attention-based Models

The most prevalent use of attention is in Transformer and BERT models [56, 75, 81]. Such models share similar architectures. In a top-down view (Figure 1), an attention-based model comprises multiple (often identically parameterized) attention blocks. An attention block comprises multiple layers: an attention layer, a normalization layer, followed by multiple (typically two) fully connected layers. Finally, each layer comprises one or more operations or operators.

2.1.2. Computation Operators

An attention layer comprises the following operators: i) Query (Q), Key (K), and Value (V)
operators that perform a projection of the input tensor, ii) Logit (L) and Attend (A) operators that compute the attention matrix and attended output respectively, and iii) Output (O) operator that performs an output projection.

Figure 1(a) shows the computational graph of an attention layer involving these operators, the input/weight/output tensor sizes for each operator, and lists the notation for symbols. We categorize them into two: (i) activation-weight operators (Q, K, V, O), which operate on activation tensors (from previous operators) and weight tensors (model parameters), and perform a GEMM computation as conventional fully connected operators (FCs), and ii) activation-activation operators (L, A), which operate on two activations from different previous operators and perform a GEMM computation.

2.2. Performance Considerations on DNN Accelerators

DNN models are often executed on special hardware platforms or accelerators for improved run time and energy efficiency. A typical DNN accelerator is composed of: (i) compute processing elements or PEs (each PE comprises a MAC unit and local scratchpad), (ii) on-chip memory that may be organized as single or multi-level hierarchies, and (iii) off-chip memory with higher capacity and lower bandwidth compared to on-chip memory. The following algorithmic and hardware considerations determine overall performance.

2.2.1. Operational Intensity: Roofline Performance. Operation intensity (Op. Int.) determines the maximum possible performance of an operator given a set of hardware resources. The operational intensity for an operator is defined as the number of arithmetic operations divided by the number of memory accesses. It captures the relative compute-memory-boundedness of an operator. A lower operational intensity implies an operator has fewer opportunities for data reuse and is more likely to be BW-bounded. This directly decides the roofline performance of the underlying accelerator on a given workload, as shown in Figure 3(a).

\[
\text{Op. Int.} = \frac{\text{num. of operations}}{\text{num. of memory accesses}} \tag{1}
\]

2.2.2. Dataflow: Achieved Performance. Dataflow refers to the mechanisms to stage data from the off-chip memory hierarchy to the compute PEs, over space and time. It determines the actual achieved performance (Figure 3(c)). Since memory access is often the bottleneck in executing DNN operators, the dataflow exposes data reuse opportunities across operands that can be exploited in hardware via buffering and data forwarding/broadcast. Formally, the dataflow encompasses: (i) tiling (how tensors are sliced, stored and fetched across the memory hierarchy), (ii) compute order (in which loop iterations are performed), and (iii) parallelism (how compute is mapped across PEs spatially). The dataflow along with specific tile sizes is often called a mapping. Examples of popular dataflow mechanisms include weight-stationary, row-stationary or output stationary.

2.2.3. Operator Fusion: Intermediate Output Reuse. Operator fusion is an optimization that schedules back-to-back operators together such that the producer’s output directly feeds the consumer, thus avoiding materialization of full intermediate tensor in memory. In state-of-the-art DNN execution frameworks, such as TensorFlow and TVM, MatMul operators (CONV or FC) are often fused with element-wise operators (such as activation functions), reshares, or shuffling operators. Fusion of multiple CONV layers has been proposed before. However, fusing multiple MatMul operators is rarely leveraged in practice within current compilers and frameworks as it has been reported to be either too complicated or unprofitable in recent work. We show in §3.1 that traditional MatMul ops often have sufficient reuse opportunities due to high op-intensity. For attention MatMuls, however, we show in §4.1 that fusion can be profitable and this opportunity has not previously been explored in literature.

3. Challenges with Running Attention Layers

In this section, we identify the challenges with achieving high compute efficiency for attention layers on DNN accelerators. As described in §2.2, achieved performance is bounded by the hardware resources and the operation intensity of the operators in the given workload (DNN model). The dataflow identifies the mechanisms to achieve performance close to this bound.

3.1. Low Operational Intensity

We start by studying the computational attributes of the operators within attention layers.

3.1.1. Activation-Weight Operators (Q/K/V/O) For Q/K/V/O operators, following the notation in Figure 1, the number of
operations is $O(BN^D)$. The number of memory accesses for the input (activations), weight (parameters), and output (activations) tensors are $O(BND)$, $O(D^2)$, $O(BND^2)$, respectively. Therefore the operational intensity is $O\left(\frac{BN^D}{BND+D+N}\right)$. We see that increasing the batch size (B) can increase the operational intensity—the same weight value can be reused by multiple activations, leading to lower BW pressure. This is a typical technique used in activation-weight operators (e.g., CONV and FC, the staple in most DNN models) as it makes better use of the scarce memory bandwidth in accelerators and enables higher utilization of the provisioned compute FLOPs, leading to improved throughput. This is qualitatively illustrated across Figure 3(a) and (b).

### 3.1.2. Activation-Activation Operators (L/A)

For L and A operators, the number of operations is $O(BN^2D)$. The number of memory access for the two input-activations and the output-activations are $O(BND)$, $O(BND)$, $O(BN^2D)$, respectively. Therefore the operational intensity is $O\left(\frac{BN^D}{2BND+D+N}\right)$. Embedding size (D) is decided by the model, and sequence length (N) is decided by the application. Furthermore, multi-head attention is an often-used variant of the attention mechanism: it leads to higher accuracy in many tasks [80]. It splits the output of the Q/K/V operator along a hidden dimension, reshaping it from size [N, D] to [H, N, d], where d=D/H. The operational intensity of L, A becomes $O\left(\frac{B^2N^2D}{2BND+D+N}\right)$. For these operators, one can not simply increase the batch size to increase the operational intensity. This is qualitatively illustrated across Figure 3(a) and (b).

### 3.1.3. Roofline Analysis

To quantitatively demonstrate the effect of operation intensity across different operators, we show the roofline analysis of operators of three common attention-based models [27, 55, 80] and a widely used CNN network ResNet50 [36] on TPU-v3 [44] in Figure 2. We can see that CONV operators scatter across both memory and compute-bound region; however, with increase in batch size, their operation intensity increases and can become compute-bound. This demonstrates why batching is a popular technique for FC layers. In contrast, L and A operators sit at memory-bound and low performance region, and batch size increase is not effective in these operators (§3.1.2). In summary, there is unfortunately little room to improve performance of L/A operators that a dataflow/mapping exploration tool can exploit because of low roofline performance.

### 3.2. Large Intermediate Tensor Footprint

Given the low operational intensity for L/A operators, one technique that can alleviate the memory bandwidth challenge is to stage intermediate tensor data on-chip to leverage the higher on-chip memory bandwidth. This is in fact done in current compiler frameworks [5, 7]. The idea is the following: since a model is being executed in an operator-by-operator (or layer-by-layer) manner, the intermediate activation tensor need not be written to off-chip memory and can be kept on-chip for the next operator to consume. While this is a promising solution for the comparatively low performance L/A operator (Figure 2), it comes with a challenge unique to L/A—namely a quadratic tensor footprint.

From Figure 1 we can calculate the intermediate tensor between L and A operators has size $O(BHN^2)$ (M-Gran in Table 1). This footprint grows quadratically with sequence length, and exceeds 8GB (exceeding the viable on-chip memory in many data-center class accelerators [6, 44]) beyond sequence lengths of 16K (Figure 3(e)). As NLP tasks with larger sequence lengths become popular [25, 50, 64], this technique is not scalable. Our proposed FLAT technique is inspired by the idea of intermediate tensor reuse, and we propose a fusion technique to allow controlling the memory footprint based on the on-chip memory constraint, as described next.

### 4. FLAT: Fused Logit Attention Tiling

We design a specialized dataflow strategy, FLAT, targeting the two memory BW-bound operators in the attention layer, L and A. FLAT includes both intra-operator dataflow and a specialized inter-operator dataflow, executing L and A in concert.

#### 4.1. Operator Fusion Opportunity

Figure 3(d) shows conceptually the potential benefit of fusing two MatMul operators. Figure 4 shows the operation intensity of single and fused operators in attention layers of a BERT model. The green dotted line marks the operation intensity threshold (ridge point) from memory to compute boundedness in TPU-v3 [44]. We observe that for FC-based operators (K/Q/V/O), the operational intensity is sufficient to be compute-bound, while for L/A it is low, as we had also observed via Figure 2). However, after fusing L and A (f(L, A)), the effective operational intensity (of the fused operator) is higher. f(FC, FC) also boosts operational intensity; however single operator FC is sufficient to achieve the performance roofline, and higher operational intensity does not bring additional performance gain. f(L/A, FC) (or f(FC, L/A)) has operational intensity in between FC and L/A, meaning fusing them can potentially lower the performance compared to the single-operator FC and L/A. This motivates us to explore L and A fusion in this work. We discuss challenges with this in §4.3.
In the compute flow of attention layers (Figure 1(a)), we decide to fuse only L and A operators and leave others unfused for the following reasons.

### 4.2. When to Fuse?

In the compute flow of attention layers (Figure 1(a)), we decide to fuse only L and A operators and leave others unfused for the following reasons.

#### 4.2.1. Fusing Other Operator Pairs

We did not fuse other operator pairs such as f(Q, L), f(A, O), or f(V, A), for three reasons. (1) The operational intensity is often enough and can be increased by leveraging batch size to reach compute-bound (Figure 2). (2) Fusing two FCs (f(FC, FC)) can achieve higher operational intensity; however since the operator is already compute-bound, there is not much value in leveraging fusion (and the additional complexity) (3) We often need fine-granularity dataflow schemes to fit fused operator tensors on-chip; however fusing two activation-weight computation (f(FC, FC)) can trade-off reuse opportunity and may in fact reduce actual achievable performance (§5.2.2). Due to these reasons, we decide to fuse only L and A operators.

#### 4.2.2. Fusing Multiple Operators

We did not fuse multiple operators such as f(L, A, O) or f(K, L, A) for two reasons. (1) Fusing L/A with FC such as f(A, O) or f(K, L) can drop the potential performance of FCs comparing to their single operator performance (Figure 4). (2) The more operators we fuse, the more data we need to stage partially on-chip. Since the on-chip memory is often extremely limited, we need to execute the fused operators at a much finer granularity, which
may lead to a degradation in achievable performance (§5.2.3). With these analysis, we decide to fuse only L and A.

4.3. Fine-grained Dataflow and Tiling
Fusing L and A operators introduces new challenges that we discuss here. §5 presents implementation details.

Challenge 1: Effectively handling large intermediate tensors that do not fit in on-chip memory. Recall that the intermediate tensor between L and A has size $O(BHN^2)$. FLAT leverages the observation that while the entire intermediate tensor can be inordinately large, slices (tiles) of this tensor are sufficient to generate slices (tiles) of the output of A. FLAT executes L and A in concert or in a fused manner. We limit the slice (tile) size of the intermediate tensor such that it can fit into the on-chip memory of a DNN accelerator: this reduced-size slice is used to generate the corresponding output slice of A. By executing portions of L and A computations entirely from the on-chip memory, FLAT enables higher effective memory bandwidth for these operators, which leads higher compute utilization and ultimately better performance.

Challenge 2: Respecting data dependencies across operators. Fusing L and A causes its unique challenge of data dependency owing to the softmax operation between them. Arbitrary inter-operator tiling will violate the dependency and causes incorrect operation. FLAT performs fine-grained tiling while respecting data dependency.

5. FLAT Dataflow Implementation
The loop nests for L and A are shown in Figure 5(a). To fuse L and A, we divide the 5-level loop nest into two: outer-loop and inner-loop, as shown in Figure 6. The outer-loops are shared across L and A. The inner-loops are unique for each operator. After fusion of two operators in the example in Figure 6 and Figure 5(b), the fused operator has two inner-loops, which we run one after another (interleaved), and iterate through the shared outer-loop. However, there are two critical constraint when fusing L and A, where we discuss our solutions in the followings: (1) constraint from data dependency (§5.1), and (2) constraint from limited on-chip memory (§5.2).

5.1. Constraints from Data Dependency
Inter-operator Data Dependency. The inner loops present a variety of tiling choices. However, these are constrained by a data-dependency between L and A: the output of L feeds into an normalization function, which subsequently feeds into A. A common choice of the normalization function is the softmax function [80] which requires a reduction along a specific dimension of the tensor before scaling individual elements. Arbitrary inter-loop tiling as employed for CONV/FC layers [9, 89], will break this dependency and lead to incorrect results.

Basic execution unit: Row-granularity. The softmax reduction is along the key dimension (Figure 5(c)): this effectively captures the relative weight of each token in the input sequence (query) against other tokens in the input (key). Therefore the basic softmax unit is a $[1, N]$ array, which requires a query of $[1, K]$ and a key of $[K, N]$. This is the finest granularity we require for the intermediate tensor slice, which will undergo a softmax operation between L and A operator (see Figure 5). We name this basic unit, row-granularity. FLAT restricts the dataflow design space to honor this row-granularity.

5.2. Constraints from On-Chip Memory
FLAT employs two levels of tiling: intra-operator tiling and inter-operator tiling. We name each tile in inter-operator tiling, a FLAT-tile. FLAT computes FLAT-tile activations from L and feeds it directly to A. FLAT-tiles, the light blue box in Figure 6(a), essentially specify how many slices of the partial intermediate tensor are calculated in one pass of the fused-operator. For example, the naive FLAT-tile, when sufficient on-chip buffer is available, is to have a FLAT-tile as large as the entire tensor, which makes the light blue box the same size as the dark blue box in Figure 6(a), which means pre-fetching entire tensor on-chip and compute the consecutive inner-loops without going off-chip. In a buffer-limited situation, we want to explore small FLAT-tiles choices, discussed later.

5.2.1. FLAT-tile and Execution Granularity
In the 5-level for loop for the L, A operators, as shown in Figure 5(a), FLAT-tile needs to include at least the bottom two levels of for loop to keep the basic unit, row-granularity. This leaves us three hyper-parameters in FLAT-tile: number of rows ($R_x$), head tile size ($H_y$), and batch tile size ($B_z$). We refer to these as Row (R-Gran), Head (H-Gran), and Batch (B-Gran) execution granularity respectively. Further, for the most intuitive baseline of moving entire intermediate tensor on-chip is referred to as Batch-Multi-Head granularity (M-Gran).

5.2.2. M-Gran, B-Gran, H-Gran: Special Characteristics of f(L, A) To fit into the limited on-chip memory, one may target to utilize finer granularity such as replacing M-Gran with B-Gran and using smaller batch size. However, note that when we are tiling two operators at finer granularity at outer loop, we may trade-off the reuse opportunity at the inner-loops. For example, for $f(FC, FC)$ and $f(CONV, CONV)$, when decreasing the batch size (i.e., micro-batching), we directly reduce the number of times a weight can be reused. The weight need to be re-fetched again and again for each micro-batch, let alone considering at even finer granularity such as H-Gran for K/Q/V/O operators. The reduced reuse opportunity by inter-operator tiling will reduce the achievable performance, leading to low achieved performance, even though the fused operator has huge operation intensity.

However, L and A perform activation-activation operation (§3.1.2). Each new activation of L needs to compute with a new activation of A, i.e., there are no activation reuse opportunity at algorithmic level. With the reduce of granularity (M-Gran to B-Gran to H-Gran), we do not lose any reuse opportunity at algorithmic level, since there are no reuse opportunity at algorithmic level even in the naive M-Gran manner. The different granularity of execution by FLAT is highly suited for $f(L, A)$ because of this special characteristic.
Table 1: Buffer requirement for tiling granularity. M: batched Multi-head, B: Batch, H: Head, R: Row.

| Granularity | M-Gran | B-Gran | H-Gran | R-Gran |
|-------------|--------|--------|--------|--------|
| Buffer Requirement | O(8BDN+BW^2) | O(8DN+H^2) | O(8Nd+RN) | O(4Rd+4Nd+RN) |

5.2.3. R-Gran: Extreme Large Sequence Range. The challenge of the nowadays system to leverage \( L \) and \( A \) fusion is that we have the need to run extreme large sequence tasks [25, 50, 64], however with limited on-chip memory resources [6, 44]. R-Gran in FLAT shows one of the potential solution for this. Nevertheless, we should understand the tradeoff. When we reduce the number of rows \( R_x \) to execute per inner-loop, we will also reduce the reuse opportunity in the matrix multiplication itself. Also, reducing number of rows at outer-loop could also decrease the achievable performance at inner-loop. e.g., not enough dimension size to fully utilize PE array. Thus, FLAT co-explore inter-operator (optimizing the outer-loop) and intra-operator dataflow (optimizing the inner-loop) to reduce the potential cause of inefficiency.

5.2.4. On-chip Buffer Requirement. Table 1 lists the required on-chip buffer size using FLAT. We derive the R-Gran value here (others follow a similar reasoning). \( L \) operator consumes \((Rd+Nd)x2\) size of on-chip buffer (2 to account for double buffering), and \( A \) consumes \((Nd+Rd)x2\). \( RN \) for buffering the intermediate tensor (FLAT-tile) (no double buffering since it does not interact with off-chip memory), whose on-chip buffer requirement is shown in Table 1.

6. FLAT Map-Space Exploration Methodology

We developed a detailed analytical cost model for estimating the performance and energy for both baseline and FLAT dataflows, as shown in Figure 7. The cost model was built similar to some previous works such as Timeloop [59], MAESTRO [4], and SCALE-Sim [68]. However, unlike these models [4, 59, 68] that design space cannot explore the design space enabled by FLAT, our cost-model can model both inter-operator and intra-operator dataflows and the mix of them, supports all the techniques involved in FLAT, and is backward compatible to MAESTRO [4] (which in turn is RTL-validated [53]).

6.1. Accelerator Assumptions

To implement FLAT, we assume a spatial accelerator (Figure 7, as described earlier in §2.2. For FLAT, the global buffer inside the accelerator needs to be fully flexible (i.e., addressable) to partition for two hierarchy levels of tiling for inter-operator and intra-operator dataflow. Many datacenter-class accelerators already employ fully programmable scratchpads [10, 44, 70] that can be programmed to create soft-partitions across the regular unfused and fused operator tiles.

6.2. Performance and Energy Model

Compute Model. We model the compute array as a collection of PEs with configurable bandwidth from/to the global on-chip buffers. The compute array can support any intra-operator dataflow (weight/input/output stationary) which is modeled by appropriately setting which tiles are stationary and which are streamed in. We also model different choices for data distribution and reduction NoCs (systolic, tree, cross-bar) which trade-off bandwidth and distribution/collection time [53, 54]. This allows us to model both systolic arrays (e.g., TPU [45]) to more flexible spatial arrays (e.g., Eyeiss_v2 [20] and MAERI [54]). We model the overhead for switching tiles (filling and draining of the array) to reflect the cold start and tailing effect. We also account for the runtime for softmax between the \( L \) and \( A \) operators.

Buffer Model. We model hierarchy of buffers. Inside each PE, we have local scratchpad for holding three elements: input, weight, and output. The global buffer is an on-chip buffer for holding intra-operator and inter-operator tiles. While the live memory footprint (current buffer requirement for staging data on-chip) is larger than the global buffer, we model the data to be partially fetched on-chip and partially fetched off-chip. Also, while input/weight/output occupies the memory, the space for the partial sum is also often an unignorable overhead, which we also precisely capture.

Memory BW Model. There are multiple units that could interact with on-chip and off-chip memory. However, the memory BW is limited. Therefore, we model the on-chip and off-chip memory as a limited shared HW resource. That is, when multiple units are requesting data from the memory and the number of data requested exceeded the memory BW, it incurs larger memory access overhead. We model detailed data transfer across the memory hierarchy. In our cost model, we track both foreground and background tasks, and all of them shared the limited on-chip/ off-chip BW.

Energy Model. Based on activity counts generated by the performance model, we leverage Accelergy [91] to estimate the energy for compute, on-chip memory and off-chip memory accesses for both the baseline and FLAT dataflows running several design-points of the dataflows. Note that FLAT does not change the total computations or the total buffer accesses to global buffer; what it changes is the number of off-chip accesses (which are orders of magnitude more expensive in
energy than on-chip [19, 76]).

6.3. Map Space Exploration (MSE) Workflow

As shown in Figure 7, our MSE takes in a workload (DNN models), HW resources configuration (PEs, buffers) as constraint, and optimization objective (e.g., compute resource utilization, throughput, energy, runtime), and outputs the optimized dataflow with its runtime and energy performance. The built cost model is leveraged internally in the MSE framework. For search method, we use exhaustive search to find the optimum point under the user-specified objective.

7. Evaluations

7.1. Evaluation Setting

Workloads. We explore different attention-based models, including BERT (BERT-base [80]), FlauBERT [56], XLM (xlm-mlm-en [55]), TransformerXL (TrXL) [27] (TrXL-large), and T5 (T5-small [65]). We explore different sequences length on these models from N=512 to N=64K [13, 50] and the future-proofing size of N=256K. We run all the models with batch size of 64.

Performance Metric. We use compute resource utilization (Util) as the primary performance metric. It shows how efficient the compute resources are used. Util=1.0 means the compute resources are fully utilized and accelerators are executing at maximum throughput.

$$\text{Util} = \frac{\text{Runtime}_{ideal}}{\text{Runtime}_{actual}}$$

, where Runtime\textsubscript{ideal} is the ideal run time of the target operations given fully utilized PEs, and Runtime\textsubscript{actual} is the run time of the operations considering all the HW overhead, including memory bandwidth, memory sizes, on-chip data movement (distribution and reduction). In MSE, we set maximizing the compute as objective. Our MSE framework allows using energy or area as metrics to optimize for as well.

Platform Resources. Based on previously proposed cloud [6, 44, 79] and edge [3, 20, 93] accelerators, we select the amount of hardware resources for cloud and edge accelerator settings as described in Figure 8, and for both of them, we assume the accelerator are running at 1GHz. In the evaluations, we make sure Special Function Unit (Figure 7) has enough FLOPs to not bottleneck the compute flow for all variants of baseline/FLAT dataflows.

Baseline Dataflow. We compare FLAT with three set of baseline dataflows, as shown in Table 2.

Comparing Environment Setup. We use the detailed cost model described in §6 to model the performance of different dataflows with different levels of platform resources: Cloud and Edge (Figure 8). We set the bit-width of the model to be 16-bits throughout our evaluation. The performance of an accelerator depends on dataflow, HW resources, and workloads. Given these three, the cost model outputs the performance report of the accelerator, including run time, compute utilization, and energy consumption.

7.2. Utilization

7.2.1. Edge Platform Resources

Recall the definition of Util from §7.1. Lower Util implies higher run time and lower throughput. As shown in Figure 9(a)-L-A-Len512, the Base dataflow has around 0.2 Util, when the buffer size is small, and has a peak Util of 0.6, when adequate buffer is provided. It demonstrates the challenge for many existing accelerators, when running attention-based models, viz., low compute utilization (low Util).

By staging data on-chip, Flex-M can potentially increase the Util, since ideally the interfacing memory BW is much higher than Base. However, when the on-chip buffer size is not adequate to house the tensors, the accelerator needs to fetch partial tensors from on-chip and the rest from off-chip. It introduces one extra pass of memory access for each of the data compared to Base, and thus has lower performance when on-chip buffer is limited. However, when the on-chip buffer is adequate, Flex-M exceeds the performance of Base, as the trend shown in Figure 9(a)-L-A-Len512. Flex-B and Flex-H are two variants with smaller granularity. They have smaller memory footprint for on-chip buffer, whose Util can approach near 0.92, when the on-chip buffer size is large. FLAT-M, FLAT-B, FLAT-H, and FLAT-Rx show that FLAT can effectively increase the cap Util, compared to their Base counterparts. The finer-grained dataflow, FLAT-Rx, which is enabled by FLAT (but cannot be leveraged by Base and Flex), can approach near 1.0 cap performance and requires much smaller on-chip buffer to reach the cap performance, which shows the effectiveness of the fine-grained execution that FLAT provides.

To show the potential performance in the full map space of intra-operator dataflow and FLAT, we run map space exploration for both of them and get their optimal points: Flex-opt and FLAT-opt. FLAT-opt always outperforms Flex-opt, as shown in Figure 9(a)-L-A-Len512 and other plots in Figure 9.

We show similar comparisons for different sequence lengths in rows 2–4 of Figure 9(a). As the sequence length grows,
the on-chip buffer requirement increases quickly (Table 1). Thus the larger granularity options can only reach low Util under limited on-chip buffer (20KB - 2GB) in the experiments. For L/A in Figure 9(a), FLAT-Rx becomes the only dataflow that can approach their cap Util region, when the sequence is longer than 64K. Note that the most optimized baseline dataflow, Flex-opt, still has low performance because it does not have the ability to leverage inter-operator and finer-grained row granularity.

Note that we consider the operator fusion for only L and A operators and keep the other operators non-fused. In Figure 9(a)-Block/Model-Len512, we observe how the effect of L/A operators are diluted when more operators are considered. Note that other operators are mostly FC/GEMM where typical single (intra-)operator dataflow is sufficient to reach high Util, and hence most of the baseline show higher performance in Block-level and Model-level performance as well. However, L and A become BW intensive when the sequence lengths are large, where the performance of L and A operator becomes dominant quickly. In Figure 9(a), we can see that FLAT can enable high Util, close to 1.0, even with large sequence length, across the hierarchy of L and A op level, block level, and model level.

7.2.2. Cloud Platform Resources In Figure 9(b), we show the evaluation under cloud platform resources with sequence lengths ranging form 4K to 256K. For the FLAT-Rx configuration, we pick larger size of Rx, since we have larger PEs array to leverage (Figure 8(a)). When the sequence length is larger than 16K, we observe that most Flex-X has Util lower than 0.4 in L/A cases. Flex-opt can find solution with near 0.8 Util when the buffer size is large and sequence length is smaller than 16K. However, it fails to find good solution when the sequence length further increases. A similar trend can also be observed at both block-level and model-level performance.

7.3. Energy Consumption

For each of the data point in Figure 9, we show their energy consumption in Figure 10. It is worth noting that high Util in Figure 9 does not directly imply good energy performance in Figure 10; however they are highly correlated. Data points with high Util usually have better memory access pattern (less off-chip memory access and more reuse) and thus cost less memory access energy, which is usually the most dominant part in the energy consumption of accelerators. In Figure 10, we observe FLAT-X and FLAT-opt, generally have lower energy consumption than Flex-X and Flex-opt. One important factor for this advantage is the saved off-chip memory accesses from inter-operator dataflow, as discussed in Figure 5.

In Figure 10(a)-L/A-Len512, we observe that some data points of FLAT-opt have larger energy consumption than the one of FLAT-X and FLAT-Rx. Since FLAT-opts are optimal points maximizing Util, which could take larger energy. This observation can also be found in Flex-X and Flex-opt cases, e.g., Flex-B and Flex-opt in Figure 10(a)-Block-Len512. However, the objective target in the map space exploration is flex-
### 7.4. Map Space Exploration

We show the entire map-space for FLAT dataflow in Figure 11. The top-left corner means reaching high utilization with the least memory footprint. For each dataflow, there are many hyper-parameters that can be tuned which becomes one unique data point in the map space plot in Figure 11. The optimal points are picked based on the objective. For example, in our study, the objective is maximum Util. Different objective could be used such as the best Util per memory footprint, leading to points in top-left corner, or the least memory footprint, leading to points in the left-most region.

### 7.5. Comparisons of Accelerators

In the previous experiments, we study the difference of dataflow by a parameter sweep across different accelerator configurations of buffer sizes. In the following, we pick two actual HW configuration design points [3, 6, 20, 44, 79, 93] for two kinds of accelerators: Edge and Cloud. We set Edge and Cloud accelerators to have 512KB and 32MB on-chip buffers, as shown in Figure 8.

#### 7.5.1. Accelerator Performance

Across Edge and Cloud configurations, Flex-opt and FLAT-opt share the same performance for K/Q/V/O and FF1/FF2. It is because in FLAT-opt, both K/Q/V/O and FF1/FF2 are treated as non-fused operators, and hence the map space for them are the same as the one in Flex-opt. In Edge configuration (Figure 12(a)), when the sequence length is 512, FLAT-opt and Flex-opt can both reach near optimal performance. However, when the sequence length increases, the performance difference becomes more significant. In the Cloud configuration (Figure 12(b)), the performance difference exaggerates, and the run time of L/A operators starts to dominate the run time performance.

Comparisons across variants of models. We compare the performance difference across multiple models. As shown in Figure 13(a), comparing to Flex-opt, FLAT-opt achieves 1.75x speedup in Edge configuration and 1.65x speedup in Cloud configuration, while reducing the energy by 44% in Edge configuration and 55% in Cloud configuration, owing to its better memory access and reuse pattern.

#### 7.5.2. BW Requirements

Effectively using the limited off-chip BW is an important factor for the scalability of the HW accelerator in the system because DNN operation is often memory-bound and the off-chip BW is often shared across different components in the system. In Figure 13(b), we show the
We show FLAT-opt's speedup and energy consumption ratio over Flex-opt.

| SeqLen | FF1/FF2 | K/Q/V/O | L/A |
|--------|---------|---------|-----|
| 4K     | 0.20    | 0.32    | 0.27|
| 16K    | 0.39    | 0.30    | 0.24|
| 64K    | 0.31    | 0.30    | 0.27|
| 256K   | 0.35    | 0.27    | 0.24|

Average FLAT-opt achieves 71% reduction in off-chip BW requirement against Flex-opt on average.

Fig. 12: End-to-end latency breakdown over different accelerators. (a) Running BERT under edge platform resources. (b) Running XLM under cloud platform resources.

| Edge | FF1/FF2 | K/Q/V/O | L/A |
|------|---------|---------|-----|
| XLM  | 1.00    | 0.40    | 0.36|
| T5   | 1.00    | 0.38    | 0.34|
| XLM  | 1.00    | 0.36    | 0.32|
| T5   | 1.00    | 0.34    | 0.30|

Average BW requirement against Flex-opt on average.

The required BW to reach utilization rate higher than 0.95 in the most BW-intensive L-A operator when running BERT, XLM under cloud platform resources.

8. Related Works

**Dataflow/ Mapping Approaches.** All DNN HW dataflow works have focused on the CONV [2, 19, 28, 30, 33, 34, 42, 46, 57, 59, 70, 73, 82, 83, 92, 95] or GEMM [45, 88] accelerator design space. Some recent works have looked into fusion of CONV operators and leveraging pipeline execution [9, 89]. Andrei et al. [41] studied operation fusion in Transformer, however they only target operation fusion between MatMul and element-wise operators. In this work, we show the benefit of fusing MatMul with MatMul in attention.

**Model-level Approaches.** At the model level, techniques such as quantization [49, 71, 94, 96], pruning [35, 67, 84, 87], and distillation [43, 69, 75, 86] are used for compressing attention-based models. Another line of research is to make fundamental change on the attention mechanism to reduce compute and memory complexity, including techniques such as local/global attention [13, 22, 61, 62, 72], learned sparsity [26, 50, 66, 77] low rank and kernel methods [24, 25, 47, 85], and others [13, 27, 64]. These techniques are orthogonal to the ideas developed in this paper. FLAT can also be leveraged in association with these techniques when deployed on DNN accelerators to further improve run time/energy performance.

**Compiler Optimization.** Fusion is a classic compiler technique widely employed in HPC [8, 29, 32, 48, 90] and ML compilers [7, 11, 18, 51]. However, ML compilers employ fusion in a limited fashion today to fuse matrix operators (FC, CONV) with element-wise ops [58] as discussed in §2.2.3.

9. Conclusion

We identify that running attention-based models with long sequences is challenging because of low reuse within certain attention operators and quadratic growth of intermediate memory footprint, both of which compound memory BW requirements. We propose FLAT, a novel dataflow for attention layers employing inter-operator fusion (the first work to in-
vestigate this for attention layers), interleaved execution and efficient tiling to enhance operational intensity and provide high compute utilization, reduced off-chip BW requirements and scalability to long sequence lengths.

References

[1] Google Open-Sources Trillion-Parameter AI Language Model Switch Transformer. https://www.infoq.com/news/2021/02/google-trillion-parameter-ai/.
[2] Nvidia deep learning accelerator. http://nvidia.org, 2017.
[3] Coral. https://coral.ai/, 2020.
[4] Maestro: An open-source infrastructure for modeling dataflows within deep learning accelerators. http://maestro.ece.gatech.edu/, 2020.
[5] Pytorch. https://pytorch.org/, 2020.
[6] Tensorflow xla. https://www.tensorflow.org/xla, 2021.
[7] Randy Allen and Ken Kennedy. Vector register allocation. IEEE Computer Architecture Letters, 4(10):1290–1317, 1992.
[8] Nvidia ampere gpu architecture tuning guide. https://docs.nvidia.com/cuda/pdf/Ampere_Tuning_Guide.pdf, 2020.
[9] David Luan, and Ilya Sutskever. Generative pretraining from pixels. arXiv preprint arXiv:1904.00965, 2019.
[10] Eunjin Baek, Dongup Kwon, and Jangwoo Kim. A multi-neural network acceleration architecture. In 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pages 940–953. IEEE, 2020.
[11] Riyadh Baghdadi, Jessica Ray, Malek Ben Romdhane, Emanuele Patil, James Laudon, Cliff Young, and David Patterson. A domain-specific supercomputer for training deep neural networks. Communications of the ACM, 63(7):67–78, 2020.
Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In International Symposium on Computer Architecture (ISCA), pages 1–12. IEEE, 2017.

Sheng-Chun Kao and Tushar Krishna. Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm. In FPGA, pages 879–882, 2017.

Ansgam Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rns: Fast autoregressive transformers with linear attention. In International Conference on Machine Learning, pages 5156–5165. PMLR, 2020.

Ken Kennedy and Kathryn S McKinley. Maximizing loop parallelism and improving data locality via loop fusion and distribution. In International Workshop on Languages and Compilers for Parallel Computing, pages 301–320. Springer, 1993.

Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. arXiv preprint arXiv:2101.01321, 2021.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The Transformer with Relative Position embeddings. In Proceedings of the Conference on Neural Information Processing Systems (NeurIPS), 2020.

Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Timeloop: Runtime extensions for rapid ai accelerator. IEEE Micro, 39(5):102–119, 2019.

Hyoungjun Cho, Dae-Won Kang, Hyungju Youn, Jungwoo Yoon, and Sung-Key Lim. Angstrom: A flexible dataflow accelerator architecture. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2020.

Hyoungjun Kwon, Angshuman Parashar, Michael Peller, and Ananda Samajdar. Dnnfusion: accelerating deep neural networks execution with advanced operator fusion. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, pages 883–898, 2021.

Angshuman Parashar, Priyanka Raina, Yukang Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucec Khaikian, Stephen W Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evolution. In 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 304–315. IEEE, 2019.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. arXiv preprint arXiv:1802.05751, 2018.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. arXiv preprint arXiv:1802.05751, 2018.

Wei Niu, Jiexiong Guan, Yanzhi Wang, Gagan Agrawal, and Bin Ren. Dnnfusion: accelerating deep neural networks execution with advanced operator fusion. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, pages 883–898, 2021.

Ansgam Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rns: Fast autoregressive transformers with linear attention. In International Conference on Machine Learning, pages 5156–5165. PMLR, 2020.

Ken Kennedy and Kathryn S McKinley. Maximizing loop parallelism and improving data locality via loop fusion and distribution. In International Workshop on Languages and Compilers for Parallel Computing, pages 301–320. Springer, 1993.

Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. arXiv preprint arXiv:2101.01321, 2021.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The Transformer with Relative Position embeddings. In Proceedings of the Conference on Neural Information Processing Systems (NeurIPS), 2020.

Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Timeloop: Runtime extensions for rapid ai accelerator. IEEE Micro, 39(5):102–119, 2019.

Hyoungjun Cho, Dae-Won Kang, Hyungju Youn, Jungwoo Yoon, and Sung-Key Lim. Angstrom: A flexible dataflow accelerator architecture. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2020.

Hyoungjun Kwon, Angshuman Parashar, Michael Peller, and Ananda Samajdar. Dnnfusion: accelerating deep neural networks execution with advanced operator fusion. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, pages 883–898, 2021.

Angshuman Parashar, Priyanka Raina, Yukang Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucec Khaikian, Stephen W Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evolution. In 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 304–315. IEEE, 2019.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. arXiv preprint arXiv:1802.05751, 2018.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. arXiv preprint arXiv:1802.05751, 2018.

Wei Niu, Jiexiong Guan, Yanzhi Wang, Gagan Agrawal, and Bin Ren. Dnnfusion: accelerating deep neural networks execution with advanced operator fusion. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, pages 883–898, 2021.

Angshuman Parashar, Priyanka Raina, Yukang Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucec Khaikian, Stephen W Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evolution. In 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 304–315. IEEE, 2019.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. arXiv preprint arXiv:1802.05751, 2018.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. arXiv preprint arXiv:1802.05751, 2018.

Wei Niu, Jiexiong Guan, Yanzhi Wang, Gagan Agrawal, and Bin Ren. Dnnfusion: accelerating deep neural networks execution with advanced operator fusion. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, pages 883–898, 2021.
[90] Michael Joseph Wolfe. *Optimizing supercompilers for supercomputers*. PhD thesis, University of Illinois at Urbana-Champaign, 1982.

[91] Yannan N. Wu, Joel S. Emer, and Vivienne Sze. Accelrys: An Architecture-Level Energy Estimation Methodology for Accelerator Designs. In *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2019.

[92] Xuan Yang et al. Interstellar: Using halide’s scheduling language to analyze dnn accelerators. In *ASPLOS*, pages 369–383, 2020.

[93] Amir Yazdanbakhsh, Kiran Seshadri, Berkin Akin, James Laudon, and Ravi Narayanaswami. An evaluation of edge tpu accelerators for convolutional neural networks. *arXiv preprint arXiv:2102.10423*, 2021.

[94] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*, 2019.

[95] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170, 2015.

[96] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812*, 2020.

[97] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.