Understanding the Performance Horizon of the Latest ML Workloads with NonGEMM Workloads

Rachid Karami
Electrical Engineering and Computer Science
University of California, Irvine
Irvine, USA
karamir@uci.edu

Sheng-Chun Kao
Mountain View, USA
chuchu40507@gmail.com

Hyoukjun Kwon
Electrical Engineering and Computer Science
University of California, Irvine
Irvine, USA
hyoukjun.kwon@uci.edu

***Abstract*—Among ML operators today, GEneralMatrix Multiplication (GEMM)-based operators are known to be key operators that build the main backbone of ML models. As their computational overhead dominates the overall execution time (e.g., 42.8% - 96.6% in our results), GEMM operators have been the prime optimization targets for fast ML inference. This led to advanced GPUs and accelerators available today, which provided significant boost in the GEMM performance compared to CPUs, aligned with the lesson from Amdahl's law. However, accelerating GEMM has significantly shifted the Amdahl's law's landscape for ML inference; due to the decreased GEMM execution time, the relative execution time of non-GEMM operators is now significant. Although the importance of non-GEMM performance is increasing, we have little knowledge about the non-GEMM performance horizon in the latest hardware platforms and models.**

Therefore, to guide non-GEMM-oriented optimizations, we conduct a thorough performance analysis of 17 widely adopted ML models in Hugging Face and Torchvision on workstation and data center platforms with/without GPUs. We discover that non-GEMM performance bottleneck is a considerable issue across all the platforms and models, accounting for 11.3% to 73.6% of total latency, on average. The challenge significantly aggravates when we apply quantization, which is a common model compression technique, due to the boosted GEMM performance and extra non-GEMM operators for dequantization and requantization. To provide insights into non-GEMM optimization targets, we demystify the most dominant non-GEMM operators for each model and deployment software. We also show that widely adopted optimizations such as operator fusion do not completely address the non-GEMM performance bottleneck, where non-GEMM operators still account for 15% to 48% of total latency. We will open-source our non-GEMM-oriented benchmark framework to facilitate research in non-GEMM optimization.

I. Introduction

The success of machine learning (ML) in various problem domains, such as computer vision (CV) [24], [25], [33], [38], [51] and natural language processing (NLP) [7], [19], [56], made ML workloads pervasive in various computing platforms from edge to cloud devices. ML model inference involves billions of multiply-and-accumulate (MAC) operations (e.g., 497 billions of MAC operations for ResNet 50 [25]). Such MAC operations originate from GEneral Matrix Multiplication (GEMM)-based operators, such as CONV2D, Linear, and BMM (batched matrix multiplication). The GEMM-based operators dominate in terms of the total execution time on CPUs, as

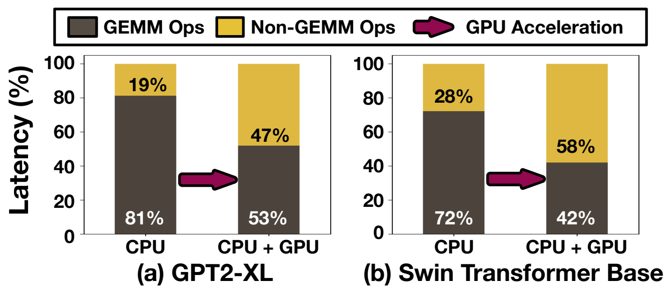


Fig. 1. The latency breakdown into GEMM and non-GEMM operators on AMD EPYC 7763 + NVIDIA A100 GPU. We measure the latency on two popular models from HuggingFace (a) GPT2-XL (batch 1) [7] and (b) Swin Transformer (batch 1) [38].

shown in Figure 1. Therefore, GPUs and accelerators have focused on the optimization of the GEMM-based operators, which significantly enhanced the computational performance (e.g., latency and throughput) of end-to-end ML model inference.

However, because the GEMM operators are being accelerated, the non-GEMM operators, such as memory operations (e.g., reshape, view, and transpose), normalization, and logit computation functions (e.g., Softmax), now account for a considerable amount of the end-to-end latency, compared to that of GEMM operators. Figure 1 shows the profiled latency breakdown into GEMM and non-GEMM operators running inferences on state-of-the-art large language (GPT2-XL [7]) and image classification (Swin Transformer [38]) models. The motivational data show that the non-GEMM operations now can account for the majority of the latency with GEMM acceleration, indicating that we now need to consider non-GEMM operators as one of the major optimization targets in the ML system optimization. However, the research community today lacks a thorough and systematic performance analysis and characterization of non-GEMM operators in the latest models, which hinders the development of non-GEMM oriented optimization techniques.

Therefore, we collect widely-used ML models from Hugging Face [59] and Torchvision [55] and perform a thorough performance characterization of of non-GEMM operators in the 17 latest models of four major task domains: Image Classification (IC), Image Segmentation (IS), Object Detection (OD), and Natural Language Processing (NLP). We evaluate the effect of GPU acceleration on the relative latency across GEMM and non-GEMM operators in collected models and conduct deep-dive analysis on the impact of different hardware platforms (workstation and data center), deployment software, and common optimizations (operator fusion and quantization). Based on our case studies, we highlight that the non-GEMM performance challenge is common in accelerated inferences and existing optimization techniques (e.g., operator fusion) cannot completely address the challenges. Also, we demystify the most time-consuming non-GEMM operators in each model, which will help the research community identify non-GEMM operators to be optimized.

To facilitate such research in non-GEMM-oriented optimization techniques, we build an open-source benchmark specialized in non-GEMM performance anlysis, NonGEMM Bench, which will be released after publication. NonGEMM Bench can profile arbitrary non-GEMM operators supported by PyTorch [44], ONNX [10], and TensorRT [4], in addition to the preset of non-GEMM operators collected from the selected 17 popular models, which provides desired flexibility to users for follow-up research.

We summarize our contributions as follows:

- We shed a light on the changed landscape of Amdhal's law in ML system design, which shows the increased importance of non-GEMM operators in systems with GEMM accelerations.
- We perform case studies on two different hardware configurations, workstation and data center, and show the non-GEMM operators are becoming a major consideration across all platforms.
- We identify different dominant non-GEMM operators depending on the model and deployment software flow, which indicates that non-GEMM optimization need to be specialized for each model and deployment software.
- We analyze the impact of a common non-GEMM-aware optimization, operator fusion and show that operator fusion does not completely mitigate non-GEMM bottleneck for all models, which motivates follow-up research in non-GEMM performance optimizations
- We evaluate the performance of non-GEMM operators with quantization and quantitatively show the non-GEMM bottleneck aggravates with quantization.
- We open-source NonGEMM Bench, an extensible benchmark flow that enables thorough non-GEMM performance characterization for any model supported by ONNX runtime, TensorRT, and PyTorch, to facilitate non-GEMM-oriented research.

II. Background

A. ML Operators

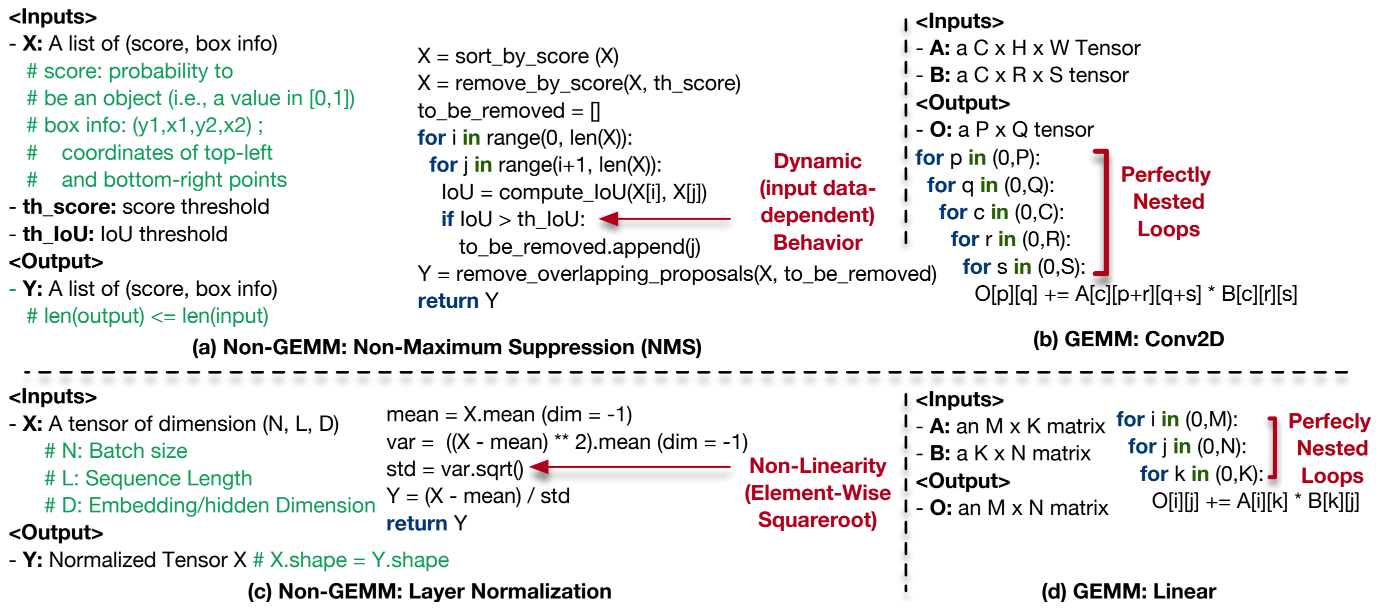
ML operators are the building blocks of ML models, which define the computation over one or multiple input tensors. Examples include convolution (Conv2d), matrix multiplication (linear, BMM, etc.), activation, and normalization, as listed in Figure 2. We categorize operators into two classes: GEMM operation-based ("GEMM operators") and the others ("Non-GEMM operators"). We discuss each class of ML operators next.

GEMM-based Operators (GEMM Operators). GEMM-based operators (or *GEMM operators*) refer to all the ML operators that can be represented as a matrix multiplication operation, which include linear, Conv2d, and batched-matrix multiplication (BMM). For example, Figure 2 (b) and (d) illustrate two popular GEMM operators: Linear and Conv2d operators, respectively. Each operator can be represented into a perfectly nested loop with multiply-and-accumulate (MAC) operation in the inner-most loop. Note that variants that are not matrix multiplication in the default form like Conv2d can be converted into GEMM (e.g., im2col [14]), which motivated the term, GEMM operator.

GEMM-based operators are known to be compute-intensive, which accounts for the majority of the execution unless accelerated by GPUs or accelerators, as CPU results in Figure 1 show. However, they have regular computation patterns that can be summarized as a perfectly nested for loop. The regular pattern allows various loop optimization techniques such as loop reordering, tiling, and parallelization, which is referred to as dataflow [34], [43], [62] With the dominance of GEMM operators in execution time and high optimization potential together, GEMM operators have been the prime optimization target for acceleration, which led to high-performance GPUs (e.g., H100 [48]) and accelerators [31].

Non-GEMM Operators. Non-GEMM operators refer to all ML operators other than GEMM operators. They span various functionalities (e.g., memory layout manipulation and normalization) other than applying weights to input tensors. Because of their diverse functionalities, their computation patterns are often not a perfectly nested loop with MAC, which can also involve non-linear functions and memory-oriented operations. For example, Figure 2 (a) shows non-maximum suppression (NMS) operator often found in R-CNN model variants [24], [50]. As found in the example, the entire operation cannot be summarized into single perfectly-nested loop, which involve other operations such as sort and filtering. In addition, the operation involves a conditional statement, which introduces non-deterministic behaviors to the operator. The layer normalization example in Figure 2 (c) also shows another key characteristic of the non-GEMM operators: non-linear functions. Because of such characteristics distinguished from GEMM operators, optimization methodologies for GEMM operators cannot be applied to accelerate non-GEMM operators.

To understand the extent of the non-GEMM operators, we analyze non-GEMM operators in 17 recent models in the computer vision and natural language processing domains. We select models based on their popularity in the Hugging Face to obtain realistic workload. We list the models we investigated in Table II. Based on our analysis, we categorize non-GEMM operators based on their functionality and summarize their usage in models and characteristics in Table I.



```
<Inputs>
- X: A list of (score, box info)
   # score: probability to
   # be an object (i.e., a value in [0,1])
   # box info: (y1,x1,y2,x2) ;
   #    coordinates of top-left
   #    and bottom-right points
- th_score: score threshold
- th_IoU: IoU threshold
<Output>
- Y: A list of (score, box info)
   # len(output) <= len(input)
```

```
X = sort_by_score (X)
X = remove_by_score(X, th_score)
to_be_removed = []
for i in range(0, len(X)):
  for j in range(i+1, len(X)):
    IoU = compute_IoU(X[i], X[j])
    if IoU > th_IoU:
      to_be_removed.append(j)
Y = remove_overlapping_proposals(X, to_be_removed)
return Y
```

Dynamic (input data-dependent) Behavior
(a) Non-GEMM: Non-Maximum Suppression (NMS)

```
<Inputs>
- A: a C x H x W Tensor
- B: a C x R x S tensor
<Output>
- O: a P x Q tensor
for p in (0,P):
  for q in (0,Q):
    for c in (0,C):
      for r in (0,R):
        for s in (0,S):
          O[p][q] += A[c][p+r][q+s] * B[c][r][s]
```

```
<Inputs>
- X: A tensor of dimension (N, L, D)
    # N: Batch size
    # L: Sequence Length
    # D: Embedding/hidden Dimension
<Output>
- Y: Normalized Tensor X # X.shape = Y.shape
```

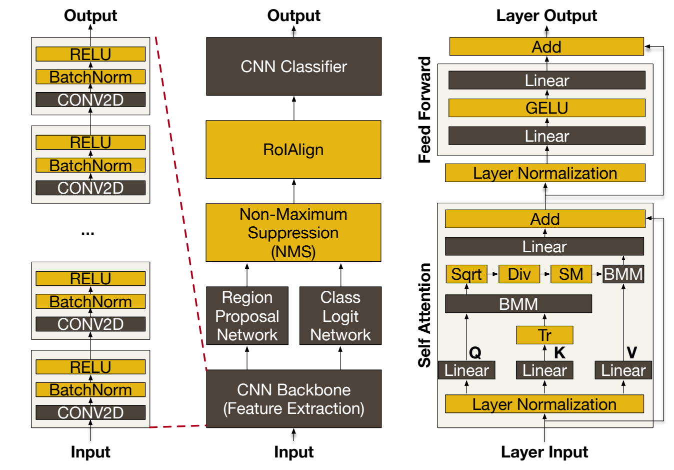
```
mean = X.mean (dim = -1)
var =  ((X - mean) ** 2).mean (dim = -1)
std = var.sqrt()
Y = (X - mean) / std
return Y
```

Non-Linearity (Element-Wise Squareroot)
(c) Non-GEMM: Layer Normalization

```
<Inputs>
- A: an M x K matrix
- B: a K x N matrix
<Output>
- O: an M x N matrix
for i in (0,M):
  for j in (0,N):
    for k in (0,K):
      O[i][j] += A[i][k] * B[k][j]
```

Fig. 2. Descriptions of example non-GEMM and GEMM operators. (a) (Non-GEMM) non-maximum suppression [24], (b) (GEMM) Conv1D (c) (Non-GEMM) Layer Normalization [9], and (d) (GEMM) Linear.

- **Normalization.** Normalization operators regularize the data range across a selected dimension using the mean and standard deviation. Examples include BatchNorm [28] and LayerNorm [9], which are widely adopted in computer vision and NLP models [7], [24]. RMS Norm [63], which is adopted in recent large language models [56], is another example of the normalization function. RMS Norm eliminates the division by standard deviation in typical normalization functions and performs $\sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(X_i - \mu)}$, where X_i, n, and μ refer to the i-th data, number of data, and the mean, respectively.
- **Activation.** Activation operators introduce non-linearity into the model. Rectified Linear Unit (ReLU) function [41] is an example of activation operators widely used in CNN based ML models [25], [51], [52]. ReLU injects non-linearity into the model based on the sign of the data by applying the element-wise function, $ReLU(X) = Max(0, X)$. Another variant of activation operators is the the Gaussian Error Linear Units function (GELU) [26], which is a popular activation function adopted in transformer based ML models [7], [33], [38], [61]. Unlike ReLU simply gates out negative values to be 0, GELU requires to compute the Cumulative Distribution Function (CDF) of a Gaussian distribution, which is often denoted as ϕ. GELU multiplies the input X by the Cumulative Distribution Function (CDF) of a Gaussian distribution (ϕ): $GELU(X) = X * \phi(X)$ [26].
- **Memory Operators.** Memory operators are responsible for the memory allocation and the layout modification of tensors. For example, Reshape modifies the shape (e.g. dimension order) of a tensor and return a new view of the tensor following the new dimension order.



(a) CNN Architecture (Image Classification) (b) RCNN Models Architecture (Object Detection) (c) GPT2 Layer Architecture (Language Processing)

Fig. 3. Architectures of three popular ML model families.

- **Element-Wise Arithmetic.** Element-wise arithmetic operators refer to all the operations applied on individual elements in a tensor (other than activations). For example, Figure 3 (c) contains an element-wise division applied to scale the elements of tensors in the attention block.
- **RoI Selection.** RoI selection operators are found in R-CNN variants. [24], [50]. They filter down bounding boxes proposed by the region proposal network (Figure 3 (b)) and align the remaining boxes to the objects detected in the image. Non-Maximum Suppression (NMS) is an example of RoI Selection, which is described in Figure 2. Given a list of scores and bounding box information, it selects bounding boxes by applying the Intersection over Union (IoU) metric.

TABLE I
Non-GEMM operators in eight selected model variants from Table II and their characteristics. Example input shapes are captured based on inferences using real datasets.

Operator Group	Operator	Model	Single Operation	Single Operand	Non Linearity	Dynamicity	Reduction	Example Input Shape
Activation	ReLu	DETR	✓	✓				[2,64,533]
	GELU	ViT-l16		✓	✓			[1, 97, 4096]
	GELU	GPT2-XL		✓	✓			[1, 8, 6400]
	SiLu	Llama-2		✓	✓			[1, 10, 11008]
Normalization	LayerNorm	Segformer		✓	✓		✓	[2, 16384, 32]
	BatchNorm2d	Segformer		✓	✓		✓	[2, 256, 128, 128]
	LlamaRMSNorm	Llama		✓	✓		✓	[1, 10, 4096]
	FrozenBatchNorm2d	MaskRCNN		✓	✓		✓	[1, 1024, 50,68]
	FrozenBatchNorm2d	DETR		✓	✓		✓	[1, 2048, 25, 34]
	LayerNorm	DETR		✓	✓		✓	[2, 850, 256]
Elmt-wise Arithmetic	Add	Segformer	✓					[2, 16384, 32]
	Mul	Llama-2	✓					[1, 10, 11008]
	Neg	Llama-2	✓					[1, 32, 10, 64]
	TrueDiv	Segformer	✓					[2, 1, 16384, 256]
	TrueDiv	GPT2-XL	✓					[1, 25, 8, 8]
Memory	Contiguous	Segformer	✓	✓				[2, 32, 128, 128]
	Contiguous	Llama-2	✓	✓				[1, 10, 32, 128]
	Permute	ViT-b16	✓	✓				[1, 768, 196]
	Permute	GPT2-XL	✓	✓				[1, 8, 25, 64]
	Split	GPT2-XL	✓	✓				[1, 8, 4800]
	View	GPT2-XL	✓	✓				[1, 8, 1600]
	Reshape	ViT-b16	✓	✓				[1, 768, 14, 14]
	Expand	ViT-b16	✓	✓				[1, 1, 768]
	Squeeze	Llama-2	✓	✓				[1, 1, 10, 128]
Logit Computation	Softmax	DETR	✓	✓	✓		✓	[1, 25, 8, 8]
	Softmax	Segformer	✓	✓	✓		✓	[2, 1, 16384, 256]
RoI Selection	NMS	MaskRCNN				✓		[4663, 4]
Interpolation	Interpolate	Segformer		✓				[2, 256, 128, 128]

B. *ML Models and Popular Tasks*

The heterogeneity in non-GEMM operators enabled ML developers to build models supporting a wide range of modalities and tasks (e.g. computer vision and NLP). As highlighted in Figure 3, computer vision (Figure 3 (a) and (b)) and NLP (Figure 3 (c)) models are characterized by distinct architectures leveraging unique combinations of GEMM and non-GEMM operators.

For example, traditional image classification models are often based on the convolutional neural network (CNN) architecture, which cascades GEMM (Conv2d) and non-GEMM (normalization and activation)operators [25], [51]. Object detection models, such as Mask R-CNN [24], often utilize CNNs for feature extraction, region proposal, and classification, as illustrated in Figure 3 (b). Unlike image classification models, they combine the CNNs with unique non-GEMM operators such as non-maximum suppression (NMS) and ROI Align to process and filter the bounding boxes for objects. On the other hand, recent language models employ the transformer architecture, which leverages the attention mechanism introduced in [57]. Transformers combine a unique set of GEMM (BMMs and Linear) and non-GEMM (normalization, memory, and element-wise arithmetic) operators, as shown in Figure 3 (c).

As we can find in the aforementioned examples, the model architectures and the combination of GEMM and non-GEMM operators are diverse. This would mean that the performance implication of non-GEMM operators would vary across models, as our motivational data presented in Figure 1 show. This motivates a thorough characterization study that investigates (1) if the non-GEMM performance challenge is pervasive across popular models and (2) how significant their implication is, under widely adopted optimization techniques (e.g., integer quantization [29] and operator fusion [42]). Therefore, we conduct a thorough case study of the non-GEMM performance horizon.

III. Performance Characterization Methodology

To understand the realistic performance landscape of the latest ML models with non-GEMM workloads, we must (1) capture operator level performances in end-to-end inferences, (2) use widely-used models by the research community and industry, (3) cover diverse task domains, and (4) use real datasets. However, ML Benchmarks available today (e.g., MLPerf [47]), unfortunately, do not satisfy all the requirements since they do not focus on the non-GEMM operators. Long-tail bench [35] identified a similar problem as this work, but it focuses on a limited set of custom kernels, which fails to represent broad task domains. Therefore, to facilitate our non-GEMM operators analysis and better-understand the impact of non-GEMM operators on system performance, we develop a new ML benchmark, NonGEMM Bench. NonGEMM Bench provides operator-level breakdown of end-to-end inference latency in the operator graph level, which enables detailed non-GEMM operator performance analysis, as we present in Section IV. To capture the performance in the latest ML workload, we select 17 highly downloaded (more than 10K downloads on average) models from HuggingFace [1] to enhance the *representativeness* of NonGEMM Bench and our analysis. We discuss the models and datasets adopted in NonGEMM Bench in detail and describe the structure of NonGEMM Bench next.

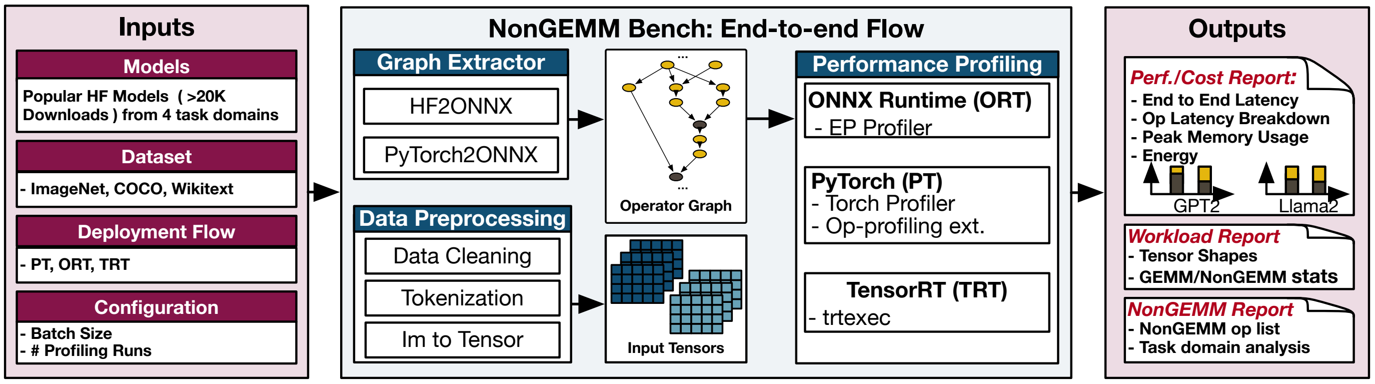


Fig. 4. An overview of NonGEMM Bench flow.

A. Models included in NonGEMM Bench

Table II lists the NonGEMM Bench model registry which contains 17 models based on state-of-the-art CNN and Transformer architectures with number of parameters ranging from 3.7M to 7B, demonstrating the *diverse* model coverage of NonGEMM Bench. The selected models cover four major task domains in ML, which include Image Classification (IC), Object Detection (OD), Image Segmentation (IS), and Natural Language Processing (NLP).

Image Classification (IC). Image classification refers to a CV task that identifies a class label of a given image. Image classification models extract features (i.e., high-level and dimensional information of the input image) from an input image and report the class label utilizing the features. NonGEMM Bench includes six most popular IC models in HuggingFace [1]: Three variants of Vision Transformer [33], and three variants of Swin Transformer [38].

Image Segmentation (IS). Image segmentation refers to a computer vision task that identifies the area in a image for each class. Like IC models, IS models also extract features and utilize them to identify objects located in an image and spatially separate them by highlighting pixels that belong to each object. NonGEMM Bench includes two state-of-the-art IS models: Segformer [61] and MaskFormer [13].

Object Detection (OD). Object detection refers to a computer vision task that identifies the location of objects in an image and outputs the bounding box of each object. OD models extract features and generate region proposals, which refer to the candidate locations and bounding boxes of objects. Using Region of Interest (RoI) processing non-GEMM operators in Table I, OD models refine raw region proposals generated by a region proposal network. The refined RoIs are used as inputs to the CNN classifier at the end, and the classifier determines the class label of objects inside each refined RoI. NonGEMM Bench includes three popular OD models [2]: FasterRCNN [50], MaskRCNN [24], and DETR [12].

Natural Language Processing (NLP). NLP refers to tasks involving the analysis and understanding of human (natural) language. NLP models extract context and features from an input text sequence and use the extracted context and features to perform multiple applications like translation and text generation [64] [11]. Transformer [57] based DNNs have become the dominant model architecture in NLP and are the backbone of popular state-of-the-art generative large language models like GPT [46] and Llama [56]. Figure 3 (c) shows the layer architecture of GPT's transformer. It consists of a self-attention block built by cascading GEMM operators with Normalization, Memory, Logit Computation and Element-wise Arithmetic non-GEMM operators (Table I). NonGEMM Bench includes six popular NLP models [3]: Bert [19], three variants of GPT2 [7], Llama2-7b [56], and Mixtral 8x7B [30].

B. NonGEMM Bench Inputs

As shown in Figure 4, NonGEMM Bench flow receives workload and dataset information (default: 17 NonGEMM Bench models in Table II), target deployment flow, and other configurations such as the batch size and number of runs for performance characterization.

Models. As described in Section III-A, NonGEMM Bench includes a registry of 17 selected popular ML models. Nonetheless, we designed NonGEMM Bench to be easily expandable to accommodate rapidly evolving ML models that constantly introduce new operators. Users can benefit from the features of our benchmark by simply plugging their new models into the NonGEMM Bench model registry (Figure 4) by specifying the model class, its weights and any data preprocessor.

Deployment Flow. NonGEMM Bench supports four popular inference frameworks: ONNX Runtime [49], PyTorch [44], TensorRT [4], and TorchInductor [8].

Datasets. To evaluate the models, NonGEMM Bench utilizes real datasets popular in each domain. We use ImageNet 2012 [17] and MS COCO [37] for computer vision tasks. As for language models, we use wikitext dataset [39] available on HuggingFace. For custom models, NonGEMM Bench allows users to specify their own dataset for their models.

Configurations. Users can specify detailed configurations for the performance characterization using NonGEMM Bench. The configurations include the batch size, the number of profiling iterations, and the target device.

TABLE II
Tasks and Models Evaluated in Section IV

Application	Models	# Params	Dataset
Image Classification (IC)	ViT base (Vt-b) [33]	307M	ImageNet [17]
	ViT large (Vt-l) [33]	307M	
	ViT huge (Vt-h) [33]	632M	
	Swin tiny (Sw-t) [38]	29M	
	Swin small (Sw-s) [38]	50M	
	Swin base (Sw-b) [38]	88M	
Object Detection (OD)	FasterRCNN (FRCNN) [50]	42M	COCO [37]
	MaskRCNN (MRCNN) [24]	44M	
	DETR [12]	41M	
Image Segmentation (IS)	Maskformer (MF) [13]	102M	COCO [37]
	SegFormer (Seg) [61]	3.7M	
Natural Language Processing (NLP)	GPT2 [7]	117M	wikitext [39]
	GPT2 Large (gpt2-l) [7]	762M	
	GPT2 X-Large (gpt2-xl) [7]	1.5B	
	Llama 2-7B [56]	7B	
	Bert [19]	110M	
	Mixtral 8x7B [30]	46.7B	

C. NonGEMM Bench Outputs

NonGEMM Bench generates many statistics organized into three categories: performance, workload, and non-GEMM-specific reports.

Performance Report. The performance report includes key performance metrics such as the end-to-end latency with operator level break-downs (Figure 6) and the end-to-end energy consumption (Figure 5).

Workload Report. The workload report includes the types of operators and the shape of the tensors for each operator captured during inference on realistic data.

Non-GEMM Report. The non-GEMM report provides insights on non-GEMM operators, such as the number of operator variants of the same class of non-GEMM operator (e.g., DETR [12] employs two variant of BatchNorm, a custom implementation and one in the PyTorch operator library) and non-GEMM operator trace on different domains.

D. NonGEMM Bench Performance Characterization Flow

NonGEMM Bench's software flow accepts inputs described in Section III-B and generates outputs described in Section III-C. Internally, NonGEMM Bench includes graph extractor, data preprocessing, and performance profiling modules.

Graph Extractor. The *Graph Extractor* module extracts the operator graphs of input models based on the selected deployment flow. NonGEMM Bench utilizes graph exporters in the HuggingFace Transformers [59] and PyTorch.

Data Preprocessing. The *Data Preprocessing* module includes model-specific preprocessing functions that fetch raw data from the target dataset, clean the data, and apply desired transformations (e.g., tokenization and image to tensor).

Performance Profiling. The *Performance Profiling* (PP) module launches the inferences, collects performance statistics, and generates output reports discussed in Section III-C. The module selects appropriate profiling functions based on the deployment flow choice. For PyTorch, the PP module utilizes the PyTorch Profiler [5]. For TensorRT [6], the PP module leverages its profiling APIs. For ONNX Runtime [49], the PP module invokes Execution Provider profiling.

TABLE III
Hardware platform configurations used for case studies.

ID	Class	CPU	GPU		
		Device	Device	Mem.	TOPS
A	Data Center	AMD EPYC 7763	Nvidia A100	80 GB	624
B	Workstation	Intel i9-13900K	Nvidia RTX 4090	24 GB	660

IV. Case Studies

We conduct a thorough performance analysis of models in Table II on a data center and a workstation-class platform, as listed in Table III. We employ PyTorch for our the main performance characterization and use ONNX Runtime, TensorRT, and TorchInductor for deep-dive studies (e.g., the impact of deployment flow choice and operator fusion).

We first focuse on the GEMM and non-GEMM performance horizon (Section IV-A). Also, we provide deeper insights into the non-GEMM performance horizon by investigating the impact of deployment flow and operator fusion (Section IV-B), and the impact of quantization (Section IV-C).

A. Non-GEMM Performance Characterization Results

We conduct a performance characterization study using PyTorch and present the results in Figure 6. Aligned with what we observed in Figure 1, the relative contribution of non-GEMM operators to the end-to-end latency significantly increases after GEMM acceleration using a GPU, from 17.2% to 42.3%, on average. However, we observe each model show different trends, mainly affected by the non-GEMM operator types and the number of GEMM and non-GEMM operators in the original model. We summarize the most time-consuming non-GEMM operators in Table IV from the data center class platform (Platform A), which shows the diversity of the dominating non-GEMM operators in each model. We highlight some notable models in each task category and delve into the details of their non-GEMM performance.

IC Task: Swin Transformer. For every Swin Transformer variant(Sw-t, -s, and -b), the memory operator group is the most latency-dominant non-GEMM operator group, which accounts for 32% of the total latency, on average, on data center platform with GPU acceleration. Those memory operators originate from the Swin Transformer's unique window shape (cross-shaped) [38], which is not well-aligned with memory layout of tensor data organized in dimension orders.

OD Task: DETR. After GPU acceleration, DETR has shown significant non-GEMM operator presence, which accounts for 65.8% of the total latency, on average. The major source of the non-GEMM latency is in the normalization operators, whose percentages are 35% and 32% on Platforms A and B, respectively. We observe the normalization functions are based on a custom implementation, which are identified as independent kernel. Kernel launch overheads accumulated for independent runs for the custom normalization function significantly contribute to the non-GEMM latency. However, we also observe that an advanced deployment software (TensorRT) can fuse those operators and significantly improve the non-GEMM performance. We discuss the details later in Section IV-B.

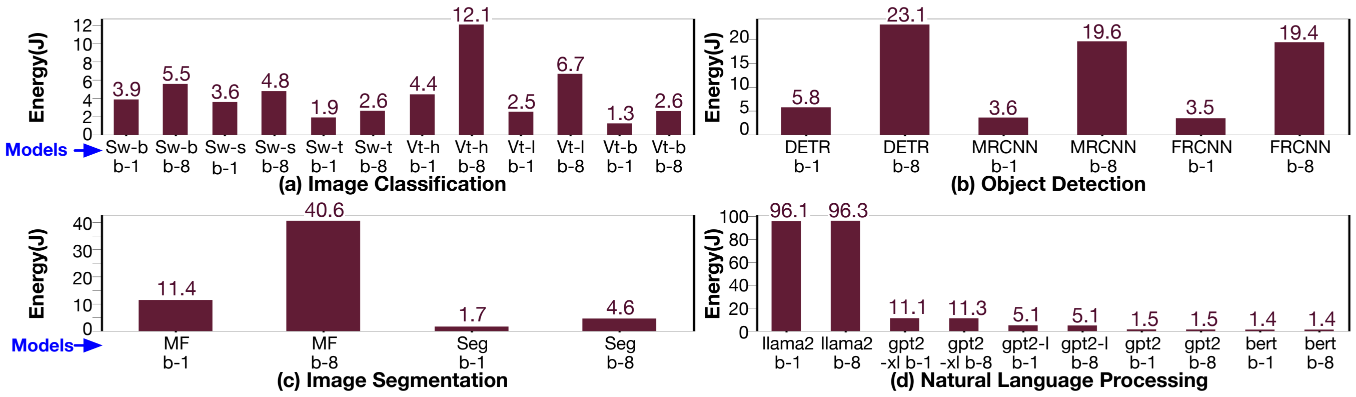


Fig. 5. End-to-End inference GPU energy consumption of models running on the Data Center (CPU + GPU) configuration.

TABLE IV
Most time-consuming non-GEMM operator groups for selected Models (Platform A, with GPU acceleration, average across batch sizes).

Task Domain	Model	Operator Group	Latency Percentage (%)
Image Classification	Vt-b	Norm	14.0
	Vt-l	Norm	13.3
	Vt-h	Norm	11.2
	Sw-t	Memory	31.8
	Sw-s	Memory	33.1
	Sw-b	Memory	32.8
Object Detection	FRCNN	Elmt-wise Arith.	34.4
	MRCNN	Elmt-wise Arith.	33.6
	DETR	Norm	34.8
Image Segmenation	MF	Memory	40.8
	Seg	Normalization	17.4
NLP	gpt2	Act	30.2
	GPT2-L	Act	29.9
	GPT2-XL	Act	28.1
	Llama2	Norm	14.9
	bert	Norm	13.1
	Mixtral	Memory	43.1

IS Task: Maskformer. MaskFormer utilizes Swin Transformer as its backbone, which introduces many memory operators as we discussed for Swin Transformer. As a result, memory operator becomes the most dominant non-GEMM operator, which accounts for 40.8% of the total latency, on average, as we can observe in Figure 6 (f).

NLP Task: GPT2. As we observe in Figure 6 (h) for both platforms, the latency of non-GEMM operators in GPT2 variants is considerable, which account for 45.0%, on average. The most dominant non-GEMM operator is an activation function, GELU, which accounts for 26.4% of the total latency.

Summary. We observe GPU acceleration significantly increases the percentage of non-GEMM operators in the end-to-end latency, which amplifies the importance of non-GEMM operators in the performance optimization process. Also, we observe the most dominant non-GEMM operators are diverse depending on the model. The results indicate that an optimization technique tailored for a single operator cannot fully address the non-GEMM performance challenge, which motivates a holistic optimization approach for wide non-GEMM operators or a balanced specialization for a set of non-GEMM operators in a target workload.

B. The Impact of Deployment Flow on Non-GEMM Performance

Deployment flows such as ONNX Runtime [49] and TensorRT [4] are widely used for serving model inferences. Such flows apply various optimizations to each model, which includes the computational graph optimizations (e.g., operator fusion) and backend assignment (e.g., utilizing Tensor Core in Nvidia GPUs). To understand the impact of deployment frameworks on the non-GEMM operator performance, we conduct two case studies: (1) comparing PyTorch and ONNX Runtime (ORT) results on LLMs (focus: general optimizations w/o operator fusion) and (2) comparing PyTorch, TorchInductor, and TensorRT results (focus: operator fusion).

[Case Study 1] Non-GEMM Performance on LLMs across PyTorch and ORT. We profile the GEMM and non-GEMM performance of two LLMs (GPT2 and Llama2) on the platform A, using the CUDA execution provider in ORT. As the results presented in Figure 7 show, we observe the presence of non-GEMM operators significantly increase, from 52.6% to a 80.75%, on average. We observe the percentage of memory operators significantly increases if we switch from PyTorch to ORT, from 3.2% to 66.8%, although the absolute end-to-end latency decreases. Such a result originates from ORT's significant performance boost of other operators (Lllma2) and ORT's limited efficiency on memory operators (GPT2-XL). Many memory operators in the evaluated LLMs are not supported by the CUDA execution provider in ORT, which leads to inefficient execution on CPUs involving costly data transfer between a CPU and a GPU. Combined with the high frequency of such operators, the relative contribution of memory operators to the end-to-end inference increases significantly, as shown in Figure 7 (b). The results imply two insights: (1) Model deployment flow can significantly aggravate the non-GEMM performance challenge and (2) the dominant non-GEMM operators differ depending on the operator support in a deployment flow.

[Case Study 2] Non-GEMM Performance with Operator Fusion. Operator fusion is one of the key optimization technique for accelerating inference workloads [4], [16], [27], [32], [42], [54]. Operator fusion combines multiple operators in a single kernel to reduce the number of costly kernel launches

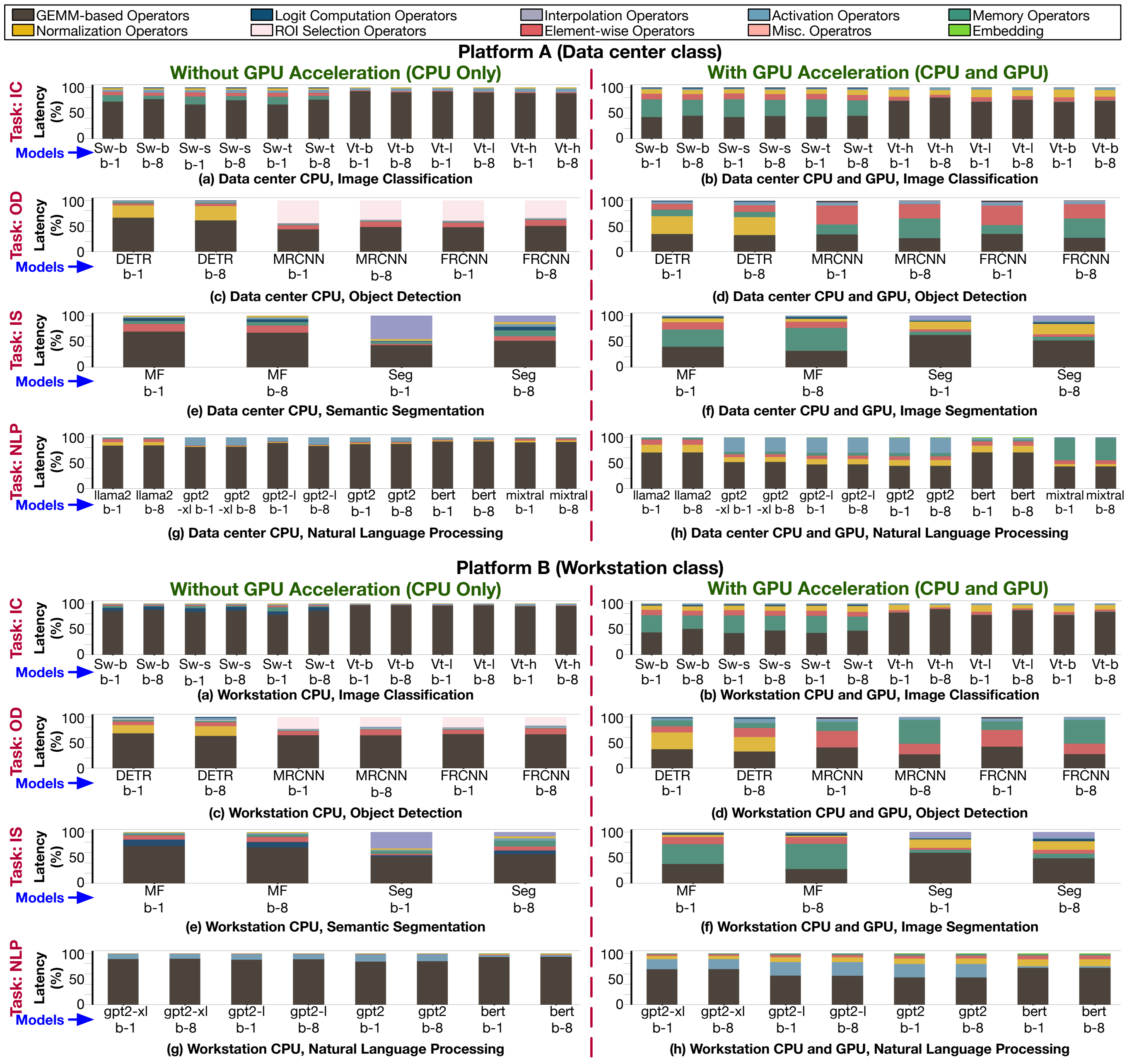


Fig. 6. Latency break-downs of NonGEMM Bench models into the operator granularity. We show CPU-only (left column) and CPU+GPU (right column) results on two evaluated platforms listed in Table III.

and minimize the number of redundant offchip memory accesses [42]. TensorRT [4] is a widely adopted inference framework released by Nvidia that applies the operator fusion technique targeting GPUs. Operator fusion in TensorRT detects specific patterns (e.g., three consecutive element-wise operators [4]) in the operator graph and fuses nodes captured in the patterns to enhance inference performance by reducing redundant memory accesses around non-GEMM operators.

To understand the impact of operator fusion on the non-GEMM performance, we conduct a case study on four models listed in Figure 8, comparing TensorRT (with fusion) and PyTorch (without fusion). We present the results in Figure 8, which shows the inference latency breakdown between GEMM and non-GEMM operators on Platform A (data center class). The results indicate that fusion mitigates the non-GEMM bottleneck, but it does not completely address the challenge. For example on Swin-b, after applying operator fusion by switching to TensorRT from PyTorch, the contribution of non-GEMM operators to the total latency changes from 56.4% to 32.2%, on average. The reduction in the percentage is based on the non-GEMM performance improvement via operator fusion, which reduces 88.6% of latency, as summarized in Table V. However, the non-GEMM operators still account for 32.2% of total latency. This shows that operator fusion cannot eliminate the non-GEMM performance challenge and motivates further studies toward non-GEMM performance optimization.

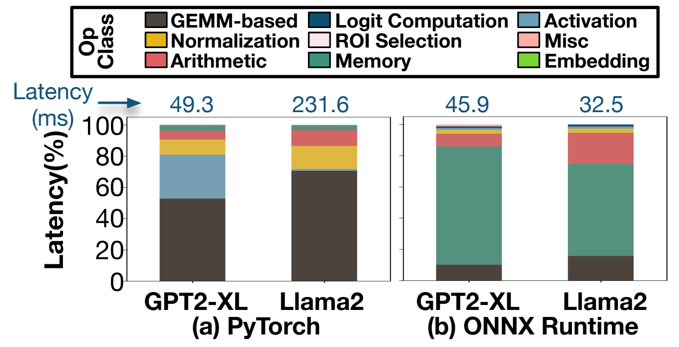


Fig. 7. The impact of deployment software toolchain into the latency breakdown on language models. (a) PyTorch and (2) ONNX Runtime on a data center class GPU (A100).

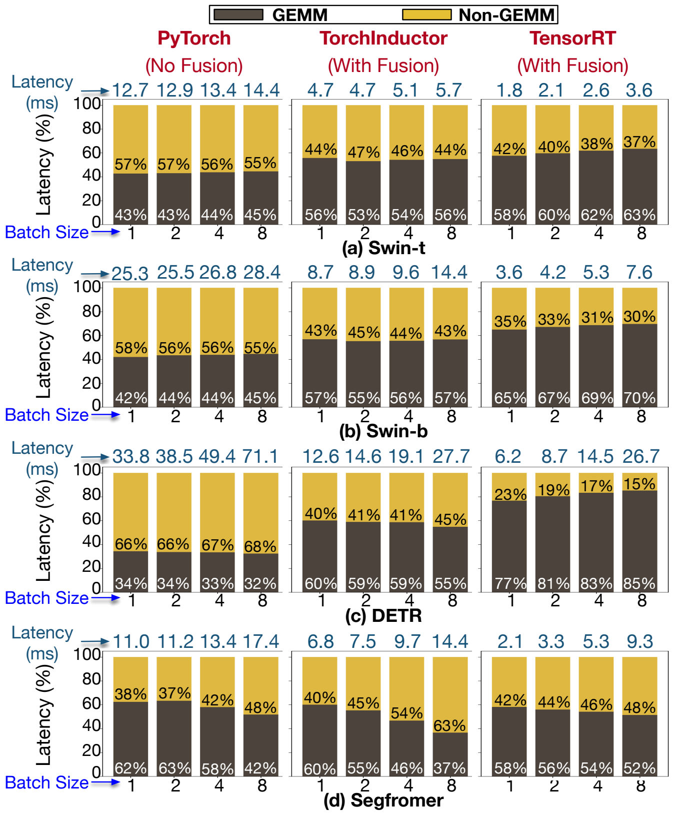


Fig. 8. The latency of non-GEMM operator in (a) Swin-t, (b) Swin-b, (c) DETR, and (d) Segformer is still considerable despite applying operator fusion in TorchInductor (middle) and TensorRT (right).

Although most results indicate a considerable impact of non-GEMM even after operator fusion, we observe that TensorRT operator fusion on the DETR model is exceptionally effective. Therefore, we conduct a deep-dive study, investigating the percentage of fused non-GEMM operators (i.e., fusion rate) and performance improvements after fusion, as listed in Table V. We observe the strong non-GEMM performance improvements for DETR originates from high fusion rate of 30%, which led to 13.5× non-GEMM speedup. This leads to the large percentage reduction of non-GEMM in the total latency, from 66.5% to 18.5%, on average.

However, the fusion rate is not the only factor that determines the non-GEMM speedup. For example, DETR and Segformer have similar fusion rates (30% and 27%,

TABLE V
The non-GEMM latency before and after applying fusion with TensorRT. The values between brackets represent the percentage with resect to the total inference latency.

Model	Non-GEMM Fusion Rate	Non-GEMM Latency	
		Before Fusion	After Fusion
Swin-t	8.8%	7.53 ms (*56.4%*)	0.97 ms (*39.0%*)
Swin-b	7.0%	14.59 ms (*56.4%*)	1.65 ms (*32.3%*)
DETR	30.0%	32.17 ms (*66.4%*)	2.38 ms (*18.5%*)
Segformer	27.0%	5.57 ms (*41.0%*)	2.33 ms (*41.0%*)

respectively), but the amount of non-GEMM performance improvements are significantly different: 13.5× and 2.39×, respectively. We analyze the execution trace and identify the fusion pattern around batch normalizations as the main source of the difference. Most batch normalization operators (100% of total) in DETR were fused with GEMM-operators (CONV+BN+ReLu pattern) while those in Segformer were fused with other non-GEMM operators (97.8% of total). The results indicate that the effectiveness of operator fusion relies on not only the overall fusion rate but also the fusion patterns. Our observation confirms that the operator fusion cannot fully address the non-GEMM performance challenge, even if it can be very effective on some patterns.

C. The Impact of Quantization Non-GEMM Performance

Quantization refers to the model optimization technique, which reduces the bit precision of model weights and/or activations to enhance the computational performance and efficiency of DNN inference. Quantization is a widely-adopted technique [29], [40] including heavy models like LLMs [18], [21], [36], [60]. LLM.int8() is a state-of-the-art quantization method, which quantizes more than 99% of the linear layers in OPT LLM to an 8-bit precision. Therefore, we adopt LLM.int8() and characterize GEMM and non-GEMM performance of Llama3 on Platform A to understand the impact of quantization on the non-GEMM operator performance problem.

As the results in Figure 9 show, we observe that non-GEMM operators dominate the latency after quantization, changing from 29.3% to 76.7%, on average. Such a significant shift in the latency distribution is mainly based on the GEMM performance improvements from 8-bit arithmetic, which reduced the latency by 38.2%, on average. However, based on our analysis on the execution traces, non-GEMM performance aggravates because the 8-bit data need to be dequantized and re-quantized for non-GEMM operators, which requires 16-bit floating point arithmetic. This introduced 6510 additional non-GEMM operators into the computation graph, which led to a significant increase of the non-GEMM latency by 5.6× after quantization. Combined together, the overall percentage of non-GEMM operators in the total latency dominate after quantization, which makes non-GEMM operators as the major optimization target.

In the case study on Llama3 8B, we observe the element-wise arithmetic operators originate from dequantization/requantization (DQRQ) process dominate in the inference latency, which adds 20% extra non-GEMM operators to

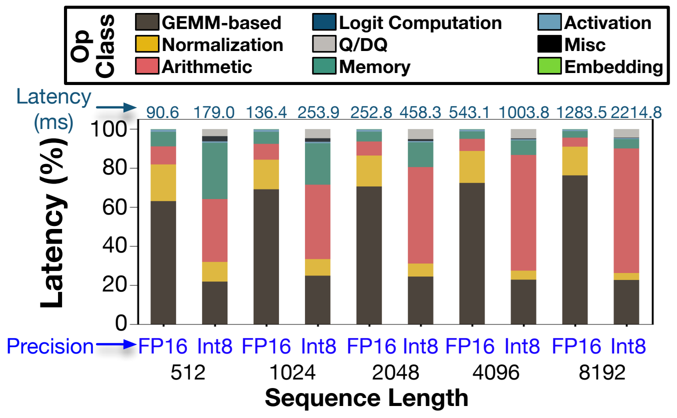


Fig. 9. GEMM and Non-GEMM latency breakdown of the inference latency on Platform A, running an 8-bit quantized Llama3 8B [18].

the original computational graph. Also, we observe longer sequence length leads to higher percentage in the element-wise arithmetic operators. For example, as we increase the sequence length from 512 to 8192, the latency percentage of element-wise arithmetic operators increase from 31.8% to 63.8%. As current trends in the LLMs are toward longer sequence lengths [7], [20], the non-GEMM performance issues in longer sequences originating from DQRQ costs will aggravate, which motivates efforts in non-GEMM performance optimization.

D. Key Observations and Insights

We summarize our main observations and insights:

- After GEMM acceleration, non-GEMM becomes a major optimization target regardless of the hardware platforms.
- Specialized optimization for one non-GEMM is not effective due to the diversity in dominant non-GEMM.
- Operator fusion cannnot fully address non-GEMM performance challenge: Although it can significantly improve non-GEMM performance, but its effectiveness heavily depends on the model.
- Operator support in deployment flows significantly affects the non-GEMM performance.
- Quantization significantly aggravates the non-GEMM performance challenge due to the imbalanced speedup across GEMM and non-GEMM and quantization/dequantization costs around non-GEMM operators

V. Related Works

ML Inference Benchmarks. Many end-to-end inference benchmarks [23], [47] do not capture operator level performance breakdowns. MLPerf Inference [47], an industry standard inference benchmark, offers a flexible and standardized framework to evaluate the performance of inference systems. MLPerf Inference framework defines performance metrics and workloads, and supports measuring the performance of realistic inference scenarios across a wide range of software and hardware systems. Nevertheless, MLPerf does not offer any operator-level fine-grained latency breakdowns, which makes it unsuitable for understanding the implications of non-GEMM operators on the inference performance. Our work provides fine-grained operator level latency breakdowns to understand the impact of non-GEMM operators on the end-to-end inference performance.

Non-GEMM Characterization. Previous works [15], [22], [35], [45], [53], [58] investigate non-GEMM operators, however their characterization and optimization focus on a limited set of operators and applications. Longtail bench [35] proposes a microbenchmark specific to a limited set of non-GEMM operators from selected computer vision models. It profiles operators without a compute library implementation in a standalone setting, using randomly generated data. Because Longtail bench is a microbenchmark suite, it cannot be used to capture realistic interplay between GEMM and non-GEMM operators in real models, which provides insights for inter-operator optimizations. In addition, Longtail bench does not provide general insights on non-GEMM performance because it focuses on a specific computer vision application. Our work extends on these efforts by studying the non-GEMM performance of 17 popular models in realistic end-to-end inference scenarios covering various task domains. Tandem Processor [22] highlights the importance of non-GEMM operator-oriented optimization in ML inference, and proposes a co-processor architecture to mitigate the non-GEMM overhead. Tandem Processor characterized the non-GEMM performance in 7 models and identify non-GEMM operators as the emerging bottleneck after accelerating GEMMs. Although Tandem is a pioneering work in non-GEMM optimization, our work provides additional and broader insights beyond it. Our work evaluates 17 widely adopted models across task domains and offers detailed case studies analyzing the impact of widely adopted optimization techniques, operator fusion and quantization, on the non-GEMM performance.

VI. Conclusion

Accelerating GEMM operators in ML inference have changed the major bottleneck from GEMM to non-GEMM operators. To understand the latest GEMM/non-GEMM performance landscape with GEMM acceleration, we conducted a thorough performance analysis of non-GEMM operators in the latest models in various task domains and platforms. The results confirm the increasing importance of non-GEMM performance and show that common model optimizations (e.g., quantization) can significantly aggravate the non-GEMM performance challenge. The dominant non-GEMM operators are diverse across models, which indicates that a specialized optimization targeting a specific operator cannot solve the non-GEMM performance challenge. We also show that non-GEMM-oriented optimization such as operator fusion cannot fully address the non-GEMM performance challenge.

Our performance anlysis results imply that now we need to consider non-GEMM operators as a major optimization target and develop new hardware and software techniques to optimize non-GEMM performance. The non-GEMM profiling software we used in this study, NonGEMM Bench, will be released as open-source software, which will contribute to the follow-up research for non-GEMM optimization.

References

[1] "Hugging face models hub," https://huggingface.co/models, accessed: 12-09-2024.

[2] "Hugging face models hub - object detection," https://huggingface.co/models?pipeline_tag=object-detection&sort=downloads, accessed: 12-09-2024.

[3] "Hugging face models hub - text generation," https://huggingface.co/models?pipeline_tag=text-generation&sort=downloads, accessed: 12-09-2024.

[4] "Nvidia tensorrt," https://developer.nvidia.com/tensorrt, 2024, accessed: 12-09-2024.

[5] "Pytorch profiler documentation," https://pytorch.org/docs/stable/profiler.html, 2024, accessed: 12-09-2024.

[6] "Tensorrt github repository," urlhttps://github.com/NVIDIA/TensorRT, 2024, accessed: 12-09-2024.

[7] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[8] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski *et al.*, "Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS)*, 2024, pp. 929–947.

[9] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[10] J. Bai, F. Lu, K. Zhang *et al.*, "Onnx: Open neural network exchange," https://github.com/onnx/onnx, 2019.

[11] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[12] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020, pp. 213–229.

[13] B. Cheng, A. G. Schwing, and A. Kirillov, "Per-pixel classification is not all you need for semantic segmentation," in *Proceedings of the 35th International Conference on Neural Information Processing Systems (NeurIPS)*. Red Hook, NY, USA: Curran Associates Inc., 2021.

[14] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[15] J. Choi, H. Li, B. Kim, S. Hwang, and J. H. Ahn, "Accelerating transformer networks through recomposing softmax layers," in *Proceedings of the 2022 IEEE International Symposium on Workload Characterization (IISWC)*, 2022, pp. 92–103.

[16] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35. Curran Associates, Inc., 2022, pp. 16 344–16 359. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/67d57c32e20fd0a7a302cb81d36e40d5-Paper-Conference.pdf

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[18] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Llm.int8(): 8-bit matrix multiplication for transformers at scale," in *Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS)*. Red Hook, NY, USA: Curran Associates Inc., 2024.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, volume 1 (long and short papers) (NAACL)*, 2019, pp. 4171–4186.

[20] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[21] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023. [Online]. Available: https://openreview.net/pdf?id=tcbBPnfwxS

[22] S. Ghodrati, S. Kinzer, H. Xu, R. Mahapatra, Y. Kim, B. H. Ahn, D. K. Wang, L. Karthikeyan, A. Yazdanbakhsh, J. Park *et al.*, "Tandem processor: Grappling with emerging operators in neural networks," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS)*, 2024, pp. 1165–1182.

[23] Y. Hao, X. Zhao, B. Bao, D. Berard, W. Constable, A. Aziz, and X. Liu, "Torchbench: Benchmarking pytorch with high api surface coverage," *arXiv preprint arXiv:2304.14226*, 2023.

[24] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2961–2969.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[26] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[27] M. Hu, A. Venkatram, S. Biswas, B. Marimuthu, B. Hou, G. Oliaro, H. Wang, L. Zheng, X. Miao, J. Zhai, and Z. Jia, "Optimal kernel orchestration for tensor programs with korch," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS)*, 2024, p. 755–769.

[28] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, pp. 448–456.

[29] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2704–2713.

[30] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[31] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[32] S.-C. Kao, S. Subramanian, G. Agrawal, A. Yazdanbakhsh, and T. Krishna, "Flat: An optimized dataflow for mitigating attention bottlenecks," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS)*, 2023, pp. 295–310.

[33] A. Kolesnikov, A. Dosovitskiy, D. Weissenborn, G. Heigold, J. Uszkoreit, L. Beyer, M. Minderer, M. Dehghani, N. Houlsby, S. Gelly, T. Unterthiner, and X. Zhai, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[34] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "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)*, 2019, pp. 754–768.

[35] X. Li, S. Yan, L. Jiang, P. Xu, J. Ma, X. Zhang, and D. Lin, "Longtail-bench: A benchmark suite for domain-specific operators in deep learning," in *Proceedings of the 2022 IEEE International Symposium on Workload Characterization (IISWC)*, 2022, pp. 282–295.

[36] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 6, 2024, pp. 87–100. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2024/file/42a452cbafa9dd64e9ba4aa95cc1ef21-Paper-Conference.pdf

[37] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Proceedings of the Computer Vision–ECCV 2014: 13th European Conference,*

Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13 (ECCV), 2014, pp. 740–755.

[38] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 10 012–10 022.

[39] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *Proceedings of International Conference on Learning Representations (ICLR)*, 2017. [Online]. Available: https://openreview.net/pdf?id=Byj72udxe

[40] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[41] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010, pp. 807–814.

[42] W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. 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 (PLDI)*, 2021, pp. 883–898.

[43] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *Proceedings of the 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315.

[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[45] S. Pati, S. Aga, M. Islam, N. Jayasena, and M. D. Sinclair, "Tale of two cs: Computation vs. communication scaling for future transformers on future hardware," in *Proceedings of the 2023 IEEE International Symposium on Workload Characterization (IISWC)*, 2023, pp. 140–153.

[46] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[47] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou *et al.*, "Mlperf inference benchmark," in *Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 446–459.

[48] NVIDIA Corporation, "Nvidia h100 tensor core gpu," 2023. [Online]. Available: https://www.nvidia.com/en-us/data-center/h100/

[49] ONNX Runtime developers, "Onnx runtime," https://onnxruntime.ai/, 2021.

[50] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf

[51] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

[52] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015. [Online]. Available: https://arxiv.org/pdf/1409.1556v6

[53] J. R. Stevens, R. Venkatesan, S. Dai, B. Khailany, and A. Raghunathan, "Softermax: Hardware/software co-design of an efficient softmax for transformers," in *Proceedings of the 2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 469–474.

[54] A. Symons, L. Mei, S. Colleman, P. Houshmand, S. Karl, and M. Verhelst, "Stream: A modeling framework for fine-grained layer fusion on multi-core dnn accelerators," in *Proceedings of the 2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2023, pp. 355–357.

[55] TorchVision maintainers and contributors, "Torchvision: Pytorch's computer vision library," https://github.com/pytorch/vision, 2016.

[56] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[57] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[58] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 97–110.

[59] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP)*, 2020, pp. 38–45.

[60] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2023, pp. 38 087–38 099.

[61] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: simple and efficient design for semantic segmentation with transformers," in *Proceedings of the 35th International Conference on Neural Information Processing Systems (NeurIPS)*. Red Hook, NY, USA: Curran Associates Inc., 2021.

[62] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina *et al.*, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, pp. 369–383.

[63] B. Zhang and R. Sennrich, *Root mean square layer normalization.* Red Hook, NY, USA: Curran Associates Inc., 2019.

[64] J. Zhu, Y. Xia, L. Wu, D. He, T. Qin, W. Zhou, H. Li, and T.-Y. Liu, "Incorporating bert into neural machine translation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. [Online]. Available: https://openreview.net/forum?id=Hyl7ygStwB

Artifact Appendix

A. Abstract

This appendix describes the workflow to run NonGEMM Bench and reproduce the results reported in the paper.

B. Artifact check-list (meta-information)

- **Algorithm**: Profiling functions are deployment flow specific. We use the PyTorch Profiler [5] for PyTorch, EP Profiling for ONNX RUNTIME [49], and TensorRT Open Source Software (OSS) for TensorRT [6].
- **Program**: Python 3.10, CUDA 12.6.
- **Model:** Please refer to Table II.
- **Data set:** Please refer to Table II.
- **Run-time environment:** Tested Environments: Ubuntu 22.04, Linux Mint 21.1, and MacOS 14.2.1.
- **Hardware:** AMD EPYC 7763 CPU, 1TB DDR4 RAM, 1 x NVIDIA A100 80GB (PCIe), Intel i9-13900K, 64 GB DDR5 RAM, and Nvidia RTX 4090 24GB (PCIe).
- **Execution:** Automated Scripts. Please refer to the README file in the Github repository.
- **Metrics:** Latency.
- **Output:** Plots in PNG format, and the corresponding data in csv format. The automated scripts plot the operator level end-to-end inference latency breakdown of all NonGEMM Bench profiled models.
- **Experiments:** Please refer to Section IV for more details.
- **How much disk space required (approximately)?:** Approximately 100 GB to store the models, the datasets, and the collected profiling traces.
- **How much time is needed to prepare workflow (approximately)?:** Approximately, setting up the workflow requires around 30 minutes.
- **How much time is needed to complete experiments (approximately)?:** Approximately 10 hours.
- **Publicly available?:** https://doi.org/10.5281/zenodo.15043135
- **Code licenses (if publicly available)?:** MIT License.
- **Archived (provide DOI)?:** 10.5281/zenodo.15043135

C. Description

1) How to access: The source code is available on Zenodo at https://doi.org/10.5281/zenodo.15043135, or on Github at https://github.com/UCI-ISA-Lab/NonGEMM-Bench-ISPASS25.git.

2) Hardware dependencies: To reproduce the paper's results, the following systems are required:

- Server with an AMD EPYC 7763 CPU, 1TB DDR4 RAM, 1x Nvidia A100 80GB GPU. (We note that the Mixture of Experts model profiling requires 4x Nvidia A100 80GB GPUs.)
- Workstation with an Intel i9-13900K, 64 GB DDR5 RAM, 1x Nvidia RTX 4090 24 GB GPU.

Nevertheless, our workflow runs on any typical laptop, workstation, or server system with a CUDA-capable GPU.

3) Software dependencies:

- Python 3.10
- CUDA 12.6
- TensorRT 10.4.0.26
- TensorRT Open Source Software
- PyTorch
- Torchvision
- ONNX Runtime
- Hugging Face Transformers
- Hugging Face Datasets
- Hugging Face Optimum
- Hugging Face Accelerate
- Matplotlib
- COCO API
- Access to Llama 2 Weights on Hugging Face
- Access to Llama 3 Weights on Hugging Face
- Access to Mixtral 8x7B Weights on Hugging Face

4) Data sets: We use three publicly available datasets highlighted in Table II: ImageNet [17], COCO [37], and wikitext [39]

5) Models: We use 17 popular pretrained models from Huggingface and Torchvision. Please refer to Table II for the detailed list.

D. Installation

1) PyTorch and ONNX Runtime Flow Software Dependency Installation:

```
> cd torch_flow
> conda create -n ng-torch python=3.10
> pip install -r requirements.txt
> ## After setting up the conda environment,
> ## Install the COCO dataset dependencies.
```

Please refer to the code base for more details.

2) TensorRT Flow Software Dependency Installation:

```
> cd trt_flow
> conda create -n ng-trt python=3.10
> pip install -r requirements.txt
```

After setting up the conda environment, please refer to the TensorRT OSS Github repository (https://github.com/NVIDIA/TensorRT/tree/release/10.4) to setup TensorRT.

E. Experiment workflow

```
> cd torch_flow
> conda activate ng-torch
> ## Set the path to ImageNet and COCO datasets in run.py
> bash run_ispass_all.sh
> cd ../onnx_flow
> bash run_ispass_all.sh
> cd ../trt_flow
> conda activate ng-trt
> ## Set the path to your TensorRT OSS
> ## installation in setup.sh
> bash run_ispass_all.sh
```

Note: Before running the experiments, environment variables and global constants should be properly set to configure the path to the datasets and to the TensorRT tools. Please refer to the README file in the codebase.

F. Evaluation and expected results

Running the `run_ispass_all.sh` scripts in every subdirectory will reproduce Figure 6, Figure 7 , Figure 8, and Figure 9.

The scripts will generate the plots and the corresponding CSV data in the `torch_flow/summary`, `onnx_flow/fig6_onnx`, and `trt_flow/fig7_trt`. The raw data is stored in `non-gemm-out` directory. The reproduced latency results are expected to be close to the results in the paper, but not an exact match because of potential differences in the hardware or software environment.

G. Experiment customization

The provided scripts run the entire evaluation presented in the paper. The users can customize their experiments by modifying the following files:

- **Modifying datasets and profiling settings:** Please modify corresponding variables in `torch_flow/run.py`.
- **Profiling new models:** Please refer to `ModelProfile` class in `torch_flow/run.py` and add the desired model to the file.

H. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- https://cTuning.org/ae