Characterizing and Understanding End-to-End Multi-modal Neural Networks on GPUs

Xiaofeng Hou[1*], *Member, IEEE,* Cheng Xu[2*], Jiacheng Liu[2], Xuehan Tang[2], Lingyu Sun[2], Chao Li[2], *Senior Member, IEEE,* Kwang-Ting Cheng[1], *Fellow, IEEE,*

Abstract—Multi-modal neural networks have become increasingly pervasive in many machine learning application domains due to their superior accuracy by fusing various modalities. However, they present many unique characteristics such as multi-stage execution, frequent synchronization and high heterogeneity, which are not well understood in the system and architecture community. In this paper, we first present and characterize a set of multi-modal neural network workloads of different sizes at inference stage. We then explore their important implications from system and architecture aspects. We hope that our work can help guide future software/hardware design and optimization for efficient inference of multi-modal DNN applications.

Index Terms—Multi-modal neural networks, computation analysis, deep learning, characterization.

1 Introduction

Multi-modal DNN applications have attracted tremendous attention in recent years for its ability to outperform traditional uni-modal networks. Through fusing information from a variety of modalities, multi-modal DNNs achieve 5% to 30% improvements compared with SOTA uni-modal networks in a wide spectrum of scenarios [1]. Thus multi-modal DNNs are being deployed on multiple platforms from servers to edge devices to handle various important tasks such as multimedia, robotics and automatic driving.

Several reasons account for the popularity of multi-modal DNN applications. First, real-world scenarios naturally possess multiple modalities [2], [3]. For example, multimedia tasks often involve modalities such as image, text and audio. Decision making in robotics depends on the results obtained from multiple sensors. Multi-modal DNNs can best utilize all the available data. Second, heterogeneous data sources start to produce increasingly massive data, which contains abundant inter-modality and cross-modality information. Multi-modal DNNs is capable of interpreting these types of information to produce better results. Third, with both software and hardware support from industry and academic [4], [5], it is able to run complex multi-modal networks both on servers and edge devices.

However, despite their superiority, multi-modal DNN applications also pose severe challenges to system and architecture designs previously applied to uni-modal networks. For example, multi-modal DNN networks present a clear three-stage execution mode (encoder-fusion-head), which causes resource under-utilization in certain stages. To understand these newly emerged issues, we build several end-to-end multi-modal DNN applications and conduct an in-depth analysis of their implications at the system and architecture level.

Multiple benchmarks have been proposed to characterize DNN applications. However, most of them only focus on uni-modal networks. MLPerf [6] is a comprehensive benchmark for measuring ML inference performance across a spectrum of use cases, while TBD [7] mainly focuses on ML training. A few architecture-oriented DNN benchmarks [8], [9] analyzed the architectural implications of uni-modal DNNs but fail to analyze multi-modal DNNs. Although MultiBench [10] analyzes multiple multi-modal networks from complexity, performance and robustness, it does not characterize multi-modal networks from the perspective of system and architecture. Besides, it takes processed data as input in many scenarios, failing to provide a complete end-to-end multi-modal framework.

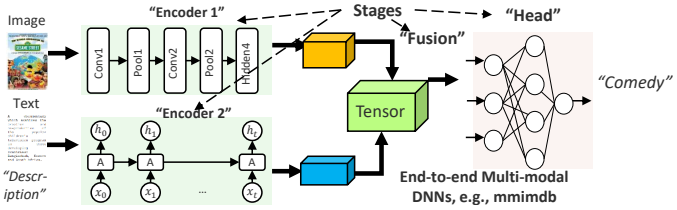


Fig. 1. The neural structure of multi-modal DNNs, which consist of three key stages including *encoder*, *fusion* and *head*.

In this paper, we first present and characterize a set of multi-modal neural network workloads of different sizes at inference stage. We implement them all in PyTorch leveraging existing model designs. We then investigate the system and architecture implications of multi-modal DNN applications.

Contributions: We first analyze the difference between different execution stages to find which stage contributes the most to the overall performance overhead. We then discover and discuss the increasing synchronization problem. Heterogeneous workload is also an important problem to be solved in multi-modal DNNs. Finally, we extend our work from GPU servers to edge devices to investigate possible bottlenecks when there are only limited resources.

2 Background

Multi-modal neural network learns and improves through the use and experience of data from multiple modalities. As depicted in Figure 1, features from multiple modalities such as image and audio are fused to produce more accurate predictions. In this work, we mainly focus on the inference stage of the applications. Compared with traditional uni-modal DNNs, multi-modal DNNs possess certain characteristics:

- *Xiaofeng Hou* and Cheng Xu* contributed equally to this work. Corresponding Author: Chao Li and Kwang-Ting Cheng.*
- *[1]ACCESS – AI Chip Center for Emerging Smart Systems, InnoHK Centers, Hong Kong Science Park, Hong Kong, China, [2]Shanghai Jiao Tong University, Shanghai, China.*

Manuscript received May 19, 2022; revised September 17, 2022.

TABLE 1
Evaluation Setup

Datasets and Workloads			
Domain	**Multimedia**		**Smart Robotics**
Application	Avmnist	Mmimdb	Mujoco Push
Model size	Small	Large	Medium
Modalties	*1.audio, 2.image*	*1.image, 2.text*	*1.control, 2.image, 3.position, 4.sensor*
Encoders	*1,2: LeNet*	*1: VGG 2: Albert*	*1,2,4: MLP 3: CNN*
Fusion methods	*Concat, tensor*	*Concat, tensor*	*Concat, tensor, transformer*
Task	Class.	Class.	Reg.
Evaluation platforms			
GPU server	NVIDIA RTX 2080TI with 11GB DDR6 NVIDIA RTX 3080TI with 12GB DDR6Besi		
Edge devices	*Jetson Nano with 128-core Maxwell and 4GB LPDDR4* *Jetson Orin with 2048-core Ampere and 32GB LPDDR5*		

Three-stage Execution. Most multi-modal DNN applications follow a three-stage execution pattern. In the first stage called *encoder*, independent neural networks translate input modalities to distinct representations suitable for machine learning. These representations are then fed to the second *fusion* stage where they are federated. Finally, the task-specific head network produces the result in the third *head* stage.

Frequent Synchronization. The fusion stage takes the representation from all modalities as input, thus requiring additional synchronization process compared with uni-modal networks. The application must wait until the completion of all modalities. Extra CPU-GPU synchronization is also needed to process intermediate data such as feature maps. CPU has to deal with data processing operations, such as *to* and *copy*.

Heterogeneous Workload. Multi-modal networks apply suitable models to process various modalities, which often involve different network structures. Besides, different fusion methods can be applied to learn from the modalities. Thus there are no universal architectural solutions to optimize all modalities with heterogeneous workloads.

3 Evaluation Setup

Benchmark. We implement three multi-modal DNN applications based on the representative datasets with different model size as shown in Table 1. These models are implemented in PyTorch with both lightweight subnets such as *LeNet*, *concat* and complex components such as *Albert* and *transformer*. Although the training process is also included, we mainly focus on profiling and characterizing the inference stage.

Profiling Platform. We evaluate our benchmark both on GPU-based server and edge devices. All the workloads are profiled on a NVIDIA 2080ti server, a NVIDIA 3080ti server, a Jetson nano board and a Jetson orin board. We use NVIDIA Nsight Systems v2021.5.2 and Nsight Compute v2022.1.1 to to analyze the GPU execution pattern as well as CPU-GPU communication. Pytorch-profiler is also applied to help analyze higher level information. Note that unless specified, we take the average value when characterizing kernels.

4 Implication on System and Architecture

Based on the setup, we investigate the characteristics of multi-modal DNNs from three main aspects: multi-stage execution, execution synchronization and workload heterogeneity, which lets us reveal the implications and suggest future optimizations.

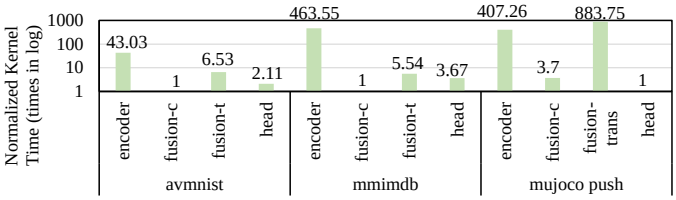
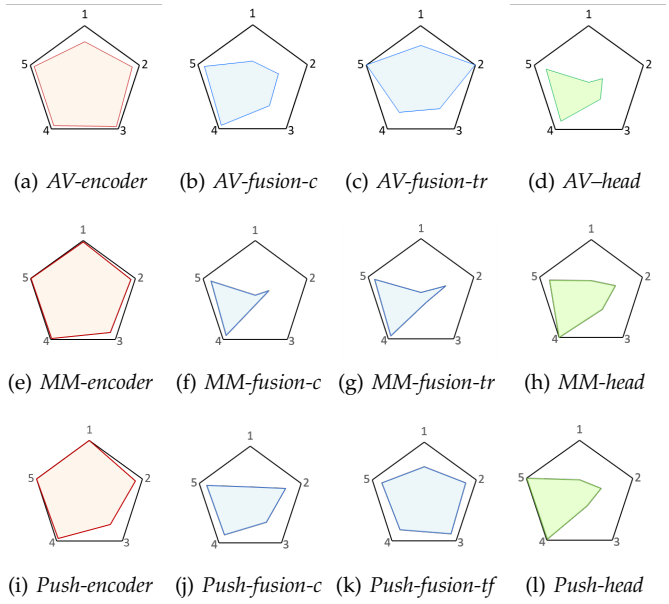


Fig. 2. Normalized GPU kernel time for the three stages. Fusion stages are implemented in different methods. *fusion-c* refers to *concat*, *fusion-t* refers to *low rank tensor*, *fusion-trans* refers to *transformer*.



(a) *AV-encoder* (b) *AV-fusion-c* (c) *AV-fusion-tr* (d) *AV–head*

(e) *MM-encoder* (f) *MM-fusion-c* (g) *MM-fusion-tr* (h) *MM-head*

(i) *Push-encoder* (j) *Push-fusion-c* (k) *Push-fusion-tf* (l) *Push-head*

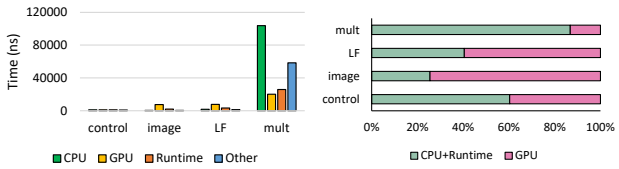
Fig. 3. Comparison of stage GPU behaviors: *1) DRAM (DRAM utilization; 2) GPU OCP (achieved occupancy); 3) IPC; 4) GLD_Eff (global memory load) efficiency; 5) GST_Eff (global memory store) efficiency. c* refers to *concat. tr* refers to *tensor*, *tf* refers to *transformer*

4.1 Stage-level Analysis

Figure 2 shows the kernel time of the three stages implemented differently on the selected datasets. Generally, *encoder* stage takes much longer time compared with *fusion* and *head* stages. This is because the fusion network takes the learned feature as input, thus having much smaller data size to deal with.

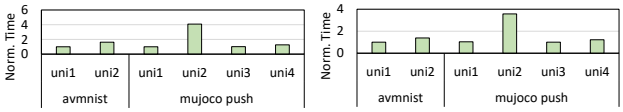
The network we adopt also matters. In the case of *mujoco push*, when we choose *transformer* as the fusion method, the kernel time of *fusion* stage becomes 2.17× of the *encoder* stage. The reason is that *transformer* gets about 15× more parameters compared with the encoders. The *encoder* stage takes the largest ratio in *mmimdb* since *VGG* and *Albert* adopted in *encoder* is more complex than *fusion* models.

We then examine the GPU resource usage of these stages in Figure 3. We take DRAM utilization (1), achieved occupancy (2), IPC (3), GLD efficiency (4) and GST efficiency (5) into consideration. DRAM utilization refers to the utilization level of GPU memory relative to peak memory usage. Achieved occupancy represents the ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor. IPC is the executed instructions per cycle. GLD efficiency means the ratio of the requested global memory load throughput to the required global memory load throughput. GST efficiency means the ratio of the requested global memory store throughput to the required global memory store throughput. In all cases, the encoder stage consumes more



(a) The time consumption of an inference task on mujoco push. (b) Comparison between synchronization and computation.

Fig. 4. Time consumption and breakdown for mujoco push. LF and Multi are two multi-modal networks implemented with different fusion methods. Control and image are counterparts of the modalities.



(a) Inference on 2080Ti (b) Inference on 3080Ti

Fig. 5. Inference time for different modalities in multi-modal DNN tasks. uni-modals are described in table 1.

resources even when fusion stage takes longer time. The most noteworthy gap exists in DRAM utilization, since fusion and head stages take learned feature as input. And the difference in GLD efficiency and GST efficiency is generally small.

Key Observations: There exists significant time and resource imbalance in different stages, which leads to possible resource under-utilization. If we assign resources to a multi-DNN application adequately to execute the encoder process, more than half of the resources, especially memory, may actually stay idle when the application enters the fusion and head stages. A possible solution is to apply more fine-grained concurrent techniques similar to DNN applications [11], ensuring QoS while improving overall system throughput.

4.2 Synchronization Analysis

Figure 4 displays the time consumption and breakdown for an inference task on GPU. There is a clear increase in CPU time and synchronization time between CPU and GPU, which means that synchronization operations outweigh computation expensive tasks on GPU. When it comes to *Multi*, the runtime cost has increased by 8.68× while the GPU time only increases by 2.38×. With much more complex network structure, CPU have to frequently gather and process information from subnets and wait for essential processing on GPU side.

Besides the synchronization of the workers, multi-modal DNNs also suffer from the problem of modality synchronization. In Figure 5, it's obvious that the execution time of different modalities is different. For example, the straggler (*uni2*) modality in *mujoco push* takes up to 4.09× of inference time compared to other modalities. When executed concurrently, such as both on deep learning accelerators (DLAs) and GPU on edge devices and in different CUDA streams on a same GPU, the fusion stage must always wait for the straggler.

Key Observations: There exists problems of frequent synchronization. A possible solution is to modify the network structure. Intermediate data such as learned representations can be stored on GPU to prevent CPUs from frequent data migration between devices. For the modality synchronization problem in the *encoder* stage, a basic approach is to use fine-grained DVFS [12] or assign less resources to manually slow down faster modalities in exchange for energy conservation.

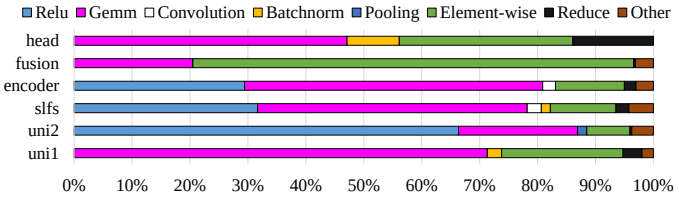
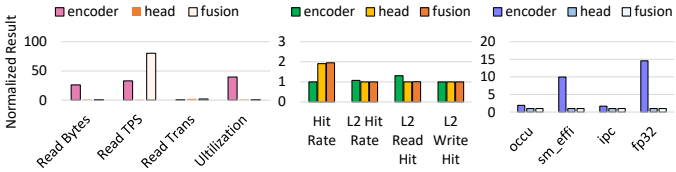
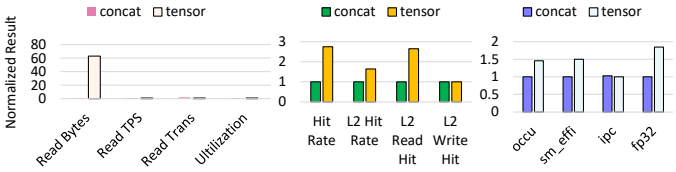


Fig. 6. Kernel time breakdown for mmimdb in different phases. *slfs* is the multi-modal implementation consisting of encoder, fusion and head. uni1 refers to image, uni2 refers to text.



(a) Hotspot kernel: *Reduce* in different stages



(b) Hotspot kernel: *Element-wise* in different fusion methods

Fig. 7. Dedicated kernel comparison different stages and fusion methods on *avmnist*. The result is normalized.

4.3 Heterogeneity Analysis

Figure 6 presents the kernel time breakdown for *mmimdb*. The proportion of different kernels of multi-modal is close to the proportion of its encoder stage, since its encoder dominates the execution time. However, there is great difference in the proportion in different stages. Fusion and head stages contain no *Relu* kernels at all, while it is one of the dominant kernels in enocder stage. What's more, the encoder stage is actually composed of two uni-modals, implemented in *VGG* and *Albert*. While *gemm* takes 70% of the total kernel time of uni0, 67% of the kernel time of uni1 is spent in *Relu*. This means that there is no universal optimization for the entire Multi-modal DNN process. Each stage, even different parts of a same stage, requires independent analysis and specific optimization methods.

Moreover, even the same kernels, perform differently in different stages and methods. We choose two hotspot kernels, *Reduce* in different stages and *Element-wise* in different fusion methods, and inspect their average resource usage in Figure 7. The resource usage of the same kernel in different fusion methods is basically at the same level despite a significant increase in DRAM read bytes. However, when it comes to the same kernel in different stages, its average resource usage can vary from 15× to 80×. The large difference in memory and compute resources possibly results from the input data size, since *fusion* and *head* only handle the learned representations from the *encoder* stage.

Key Observations: There exists heterogeneous workloads in different stages. In this regard, it is hard to find a universal optimization for the whole application. Multi-modal applications must be analyzed first to identify the bottlenecks. It is hard to design specialized hardware accelerators for multi-modal DNN applications. For example, although the time breakdown of multi-modal DNN performs similar like that of one of

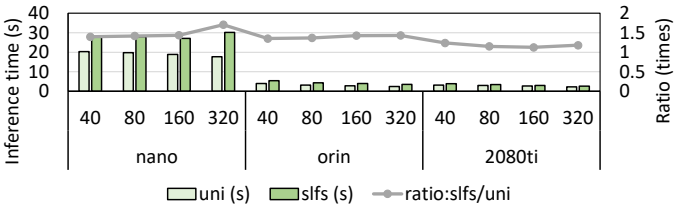
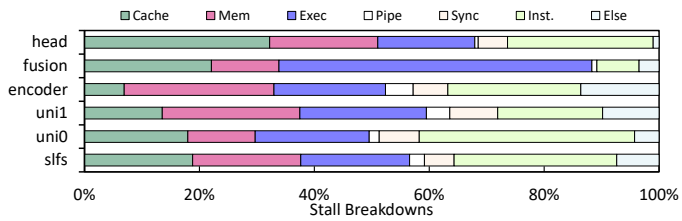
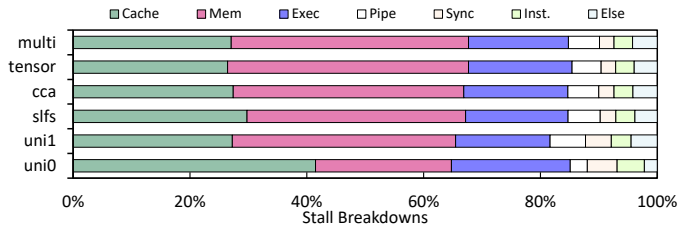


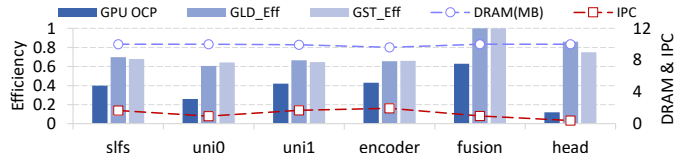
Fig. 8. Inference time of avmnist on GPU server and edge devices with the change of batch size. slfs refers to an implementation of multi-modal with 31x parameters. *uni* is one representative modality.



(a) Breakdown of stall cycles on Jetson Nano



(b) Breakdown of stall cycles on 2080ti



(c) Computation and memory usage on Jetson Nano

Fig. 9. Execution stall breakdown and resource usage on edge devices. uni0 refers to audio, uni1 refers to image.

its subnets implementing *transformer*, specific accelerator on *transformer* [13] does not apply to all stages in this application.

5 Extending to Edge Devices

Besides GPU servers, we also study the characteristics of multi-modal DNNs on edge devices since some of these applications such as robotics are generally executed on edge devices.

Figure 8 presents the inference time of *avmnist* on both GPU servers and edge devices. On Jetson nano where resources are limited, 6.48× more time is needed. With the increase of batch size, while the latency of GPU server is constantly decreasing, the latency of Jetson nano is even higher when batch size reaches 320, where certain resources are used up. On Jetson orin with abundant resources, it acts more like GPU servers. The ratio of the time of multi-modal compared with uni modal is higher on both Jetson nano and orin, since 2080ti possesses more idle resources.

In Figure 9-(a) and (b), we illustrate the execution stall breakdown and resource usage patterns of multi-modal DNN running at the edge and on GPU servers. The stall caused by execution dependency and instruction not fetch increases dramatically, while memory and cache are the main causes of stall on GPU servers. It possibly results from the lack of computing power so that requisite operations cannot be finished in time. As shown in Figure 9-(c), on edge devices with limited resources, DRAM utilization is almost always kept at the highest level. Unlike GPU servers in Figure 3, fusion stage now exhibits higher occupancy.

Key Observations: Extending to edge devices leads to higher latency and new bottlenecks. Due to limited power and resources, the inference time grows dramatically when we switch from uni-modal DNN to multi-modal DNNs even on a small dataset. It would be a huge challenge to streamline multi-modal parameters to enable them to run on edge devices. A technique called early exit can be applied to alleviate the problem, where some of the modalities are skipped as long as the result of some important modalities meets the QoS.

6 Conclusion

In this paper, we systematically study the characteristics of multi-modal DNNs on GPUs. We characterize a set of representative multi-modal DNN applications of different sizes at inference stage and quantify the implications of different execution stages, execution synchronization, and the workload heterogeneity. We further extend our work to edge devices. Our analysis suggests that multi-modal DNNs possess unique characteristics compared with traditional DNN workloads, and should be handled with customized optimizations.

Acknowledgments

This research was partially supported by ACCESS - AI Chip Center for Emerging Smart Systems, InnoHK funding, Hong Kong SAR. It is also supported in part by the National Natural Science Foundation of China (No.62122053), and Shanghai S&T Committee Rising-Star Program (No.21QA1404400).

References

[1] J. Arevalo *et al.*, "Gated multimodal units for information fusion," in *International Conference on Learning Representations (ICLR)*, 2017.
[2] C. Zhang *et al.*, "Multimodal intelligence: Representation learning, information fusion, and applications," in *IEEE Journal of Selected Topics in Signal Processing*, 2020.
[3] T. Baltrušaitis *et al.*, "Multimodal machine learning: A survey and taxonomy," in *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2018.
[4] Google, "Tensorflow: An end-to-end open source machine learning platform," https://www.tensorflow.org/, 2022.
[5] Amazon, "Pytorch: An open source machine learning framework that accelerates the path from research prototyping to production deployment," https://pytorch.org/, 2022.
[6] V. Reddi *et al.*, "Mlperf inference benchmark," in *International Symposium on Computer Architecture (ISCA)*, 2020.
[7] H. Zhu *et al.*, "Tbd: Benchmarking and analyzing deep neural network training," *IEEE/CVF Computer Vision and Pattern Recognition (CVPR)*, 2018.
[8] Q. Zhang *et al.*, "A comprehensive benchmark of deep learning libraries on mobile devices," *ACM Web Conference (WWW)*, 2022.
[9] M. Almeida *et al.*, "Embench: Quantifying performance variations of deep neural networks across modern commodity devices," *Embedded and Mobile Deep Learning (EMDL)*, 2019.
[10] P. Liang *et al.*, "Multibench: Multiscale benchmarks for multimodal representation learning," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
[11] M. Han *et al.*, "Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.
[12] X. Hou *et al.*, "Ant-man: Towards agile power management in the microservice era," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.
[13] J. Fang *et al.*, "Turbotransformers: An efficient gpu serving system for transformer models," in *Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2021.