Beyond Inference: Performance Analysis of DNN Server Overheads for Computer Vision

Ahmed F. AbouElhamayed
afa55@cornell.edu
Cornell University
New York, USA

Susanne Balle
susanne.balle@intel.com
Intel
Massachusetts, USA

Deshanand Singh
deshanand.singh@intel.com
Intel
Ontario, Canada

Mohamed S. Abdelfattah
mohamed@cornell.edu
Cornell University
New York, USA

ABSTRACT

Deep neural network (DNN) inference has become an important part of many data-center workloads. This has prompted focused efforts to design ever-faster deep learning accelerators such as GPUs and TPUs. However, an end-to-end DNN-based vision application contains more than just DNN inference, including input decompression, resizing, sampling, normalization, and data transfer. In this paper, we perform a thorough evaluation of computer vision inference requests performed on a throughput-optimized serving system. We quantify the performance impact of server overheads such as data movement, preprocessing, and message brokers between two DNNs producing outputs at different rates. Our empirical analysis encompasses many computer vision tasks including image classification, segmentation, detection, depth-estimation, and more complex processing pipelines with multiple DNNs. Our results consistently demonstrate that end-to-end application performance can easily be dominated by data processing and data movement functions (up to 56% of end-to-end latency in a medium-sized image, and ∼ 80% impact on system throughput in a large image), even though these functions have been conventionally overlooked in deep learning system design. Our work identifies important performance bottlenecks in different application scenarios, achieves 2.25× better throughput compared to prior work, and paves the way for more holistic deep learning system design.

ACM Reference Format:
AbouElhamayed, et al.. 2024. Beyond Inference: Performance Analysis of DNN Server Overheads for Computer Vision. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655960

1 INTRODUCTION

Artificial Intelligence (AI) has quickly proliferated different aspects of computing as exemplified by the rise of large language models (LLMs) within chat bots like ChatGPT—the fastest growing consumer application in history [5]. Such deep neural networks (DNNs) need to handle millions or billions of real-time customer queries on a daily basis, requiring fast, scalable, and efficient inference systems. This is equally true for computer vision applications. For instance, social media platforms like Facebook process more than 10.5 billion photos per month [4] using DNNs for person detection, automatic tagging, content classification, and style transfer. Furthermore, video broadcasting platforms such as YouTube use DNNs to automatically detect age-restricted content in addition to other tasks like auto-captioning [1]. Many of these computer vision tasks require fast and throughput-oriented inference servers to perform efficient DNN inference. In addition, many of these tasks contain non-trivial preprocessing and postprocessing functions to compress and manipulate the DNN input data for both efficiency and compatibility with the DNN. These functions, referred to as *DNN inference overheads*, are particularly common in applications involving image and video data which is large, high-dimensional, and comes in many different sizes, formats, and properties.

To fulfill an inference request, a trained DNN is deployed on a web server. These modern servers have begun to include dedicated hardware for inference, including Graphical Processing units (GPUs), and custom ASICs like Google's Tensor Processing Unit (TPU) [15], Amazon's Inferentia [19], Meta's MTIA [14], in addition to academic deep learning accelerators [7]. These accelerators have been immensely successful at accelerating the massively-parallel matrix multiplications that are present in DNN inference, however, other data manipulation tasks such as decompression, resizing, decoding, and sampling often need to *fall-back* to the host CPU for preprocessing. For example, a video classification service receives the video in a compressed format like MPEG, decodes the video, samples a number of frames, then resizes and normalizes the resulting images into the format required by the DNN. The current emphasis on enhancing DNN performance while disregarding other tasks will eventually limit performance according to Amdahl's Law [9]. This motivates a deeper investigation into the performance bottlenecks and latency breakdowns of deep-learning vision tasks on modern server systems.

In this work, we quantify the performance impact of DNN inference overheads for computer vision applications running on state-of-the-art inference systems to better understand and optimize end-to-end system performance. On an optimized serving system, we implement and profile a number of computer vision applications including image classification, segmentation, object detection, depth-estimation, and a multi-DNN application with message brokers. In addition to highlighting the impact of data processing, we study the performance and energy efficiency of different server configurations, different data processing hardware options, and implications of multi-GPU systems. Our work advances our understanding of DNN inference systems, quantifies performance bottlenecks, and advocates for holistic end-to-end DNN inference optimization.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3655960

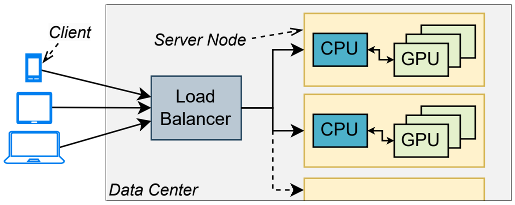

Figure 1: Sample API system serving a DNN on a GPU.

Our key contributions are:

(1) Quantify the latency, throughput, and energy efficiency implications of DNN inference overheads within deep-learning based computer vision applications. We find that non-DNN data processing can take up to 56% of the time it takes to process a DNN inference request in a medium-sized image, even within a highly optimized inference server.

(2) Evaluate different hardware systems for DNN serving systems, including CPU-GPU, GPU-only, and CPU-multi-GPU systems to investigate scaling. We draw many conclusions about optimized system setup and CPU-GPU ratios.

(3) Investigate and optimize a computer vision pipeline with two DNN inference calls (detection then identification), demonstrating 2.25× higher throughput compared to prior work.

2 BACKGROUND

2.1 Inference Systems

A common approach to deploying DNNs in various applications involves server-based execution, accessible to client devices (such as mobile phones or laptops) through an Application Programming Interface (API). In this model, a *load balancer* within the datacenter receives incoming requests and strategically distributes them among the available processing servers, as illustrated in Fig 1. Each server handles multiple concurrent requests, necessitating the implementation of multi-threading and concurrency control strategies. Our study focuses on a scenario where the load balancer imposes a cap on the number of concurrent requests each server can handle. In instances where incoming requests exceed the system's predefined capacity, additional servers are added to maintain performance.

To efficiently process DNNs, GPUs or custom ASICS (e.g. TPUs) have become necessary in modern datacenters. These devices are optimized for batch processing, thereby presenting a challenge to a server's typical individual request handling. This is a key reason for creating specialized DNN inference serving software such as NVIDIA's Triton Inference Server (TrIS). Key to its operation is *dynamic batching*, which aggregates incoming requests for batch processing often while ensuring bounded latency. Serving software provides many adjustable settings, including the maximum queuing latency, and maximum batch size. Additionally, multiple *instances*[1] of the processing units can each handle requests independently, which in turn increases the number of requests the server can handle at a time.

There are two key measures of server performance. The first is throughput that quantifies the number of requests that can be processed per second. The second is latency, to measure how long each request takes. Such servers typically produce a distribution of latencies as a result of differing arrival times, dynamic batches, and CPU load, therefore, average and tail latency are typically reported to represent the typical and worst-case server performance respectively.

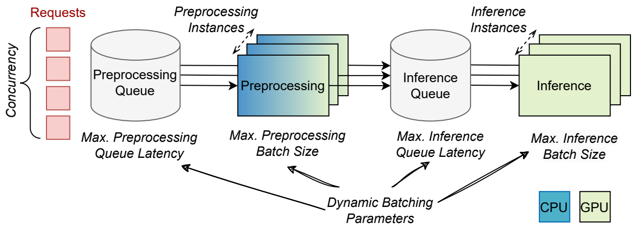
Requests
Concurrency
Preprocessing Queue
Preprocessing Instances
Preprocessing
Inference Queue
Inference Instances
Inference
Max. Preprocessing Queue Latency
Max. Preprocessing Batch Size
Max. Inference Queue Latency
Max. Inference Batch Size
Dynamic Batching Parameters
CPU
GPU

Figure 2: A sample DNN application consisting of preprocessing and inference with annotated server parameters.

2.2 Application Scenarios

A typical DNN inference pipeline is demonstrated in Fig. 2. In addition to a DNN, this system contains a preprocessing stage which transforms user data to the correct size and format required by the DNN. Typically, general preprocessing is handled by a server CPU, while the DNN inference is offloaded to an accelerator such as the GPU. However, the increase in GPU speed and stagnation in CPU performance has necessitated accelerated preprocessing solutions. This is why common preprocessing functions, especially those related to image and video data, are being accelerated on GPUs through software libraries like NVIDIA DALI [6]. The importance of preprocessing performance is further underscored by the inclusion of a dedicated hardware JPEG decoder specifically for DNN preprocessing on modern GPUs such as NVIDIA A100 [2].

Fig. 2 demonstrates a two-stage application pipeline in which both the first and second stages are *rate-matched.* However, in other scenarios one input to stage 1 can produce multiple outputs that need to be processed by stage 2. The example that we study further in Section 4.7 consists of a face detection pipeline followed by identification. In this case, a single frame could contain multiple faces to be processed by the face identification stage. In this case, typically a *message broker*—such as Apache Kafka or Redis—is used between the producer and consumer to manage communication between the two processing stages.

2.3 Software Configuration Impact

To accurately quantify DNN inference overheads, we must do so under optimized server configurations. Fig 3 highlights the stark performance differences on the same hardware platform while applying different optimizations. On a CPU-GPU system[2], we measure the performance of an image classification pipeline with the Vision Transformer (ViT) base model. First we start with the PyTorch model downloaded directly from HuggingFace and we run it without any serving software, just a Python loop that decompresses JPEG images one-by-one, followed by batched DNN inference, yielding ~431 img/s. Next, throughput increases to ~446 img/s when we use NVIDIA's DALI framework that enables batched image decompression. When enabling GPU for preprocessing which uses the NvJPEG library, the throughput increases to ~842 img/s. Employing

[1] Processes in case of CPU and CUDA Streams in case of GPU.

[2] Our setup is a dedicated server node with 13th Gen Intel(R) Core(TM) i9-13900K CPU and an NVIDIA GeForce RTX 4090 GPU.

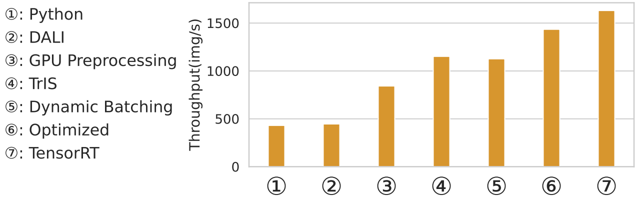

Figure 3: Evaluation of throughput across diverse system setups running the same Vision Transformer (ViT) model.

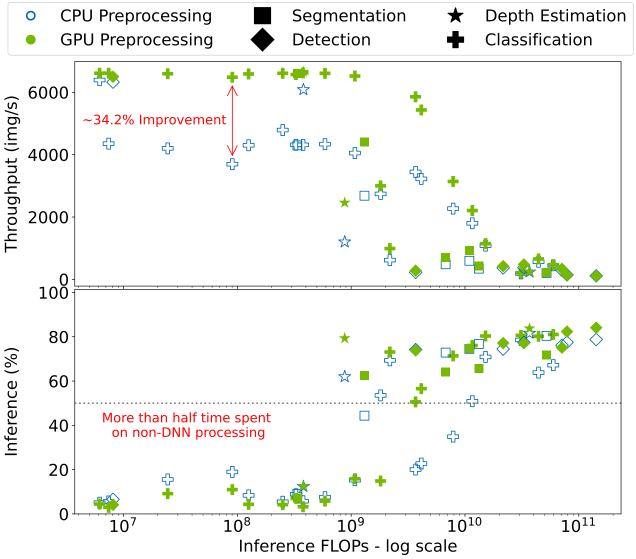

Figure 4: Throughput and inference time percentage for various HuggingFace models for both CPU/GPU preprocessing.

TrIS (with the ONNX runtime) instead of PyTorch further improves performance thanks to its more optimized model execution and asynchronous processing, allowing an overlap between computation and data movement. Next, we enable TrIS dynamic batching instead of a fixed batch size to mimic a realistic server workload. Even though throughput drops slightly, the tail latency improves from 55 ms to 38 ms, providing a better quality of service. To optimize the server setup, we perform a quick search on its settings that include the number of preprocessing and inference processes, the maximum allowed batch size, and the concurrency per server. This results in a ~300 img/s throughput improvement, showing that server software parameters are critical to high performance. Finally, we use TensorRT to enable model-level optimizations while compiling ViT, resulting in additional throughput improvement to more than 1600 img/s. Overall, there is more than 8× throughput improvement between the baseline PyTorch and optimized TrIS+TensorRT implementations. This highlights the importance of optimizing the server setup to measure realistic and representative system performance—we use this optimized setup for all our results in this paper.

3 RELATED WORK

MLPerf [16] has been a community-driven effort to standardize the benchmarking for AI workloads on different hardware platforms. Other notable efforts include Alibaba’s AI Matrix [3], Fathom [8] and DAWNBench [12]. Benchmarking end-to-end performance in this way enables a fair comparison of complete AI deployment solutions, including both the hardware system and the software stack. However, it does not expose specific performance bottlenecks within the AI workload, nor does it specifically attempt to alleviate these bottlenecks through optimized implementations. To address this shortcoming, inferBench [18], focuses on server-side inference and compares different serving frameworks including Tensorflow Serving, ONNX runtime, and TrIS, and different *serving formats* such as ONNX, TorchScript, and TensorRT. Another work, iBench [10, 11], focuses on evaluating client-side preprocessing and server-side inference using a custom Flask-based server. We build upon the findings from these prior benchmarking efforts but focus specifically in identifying the performance bottlenecks within state-of-the-art DNN servers for a number of AI computer vision applications. Finally, the term “AI Tax" was coined by Richins et al. [17], where they studied an AI image processing pipeline with face detection followed by face identification with an Apache Kafka broker in between the two stages to manage data communication. Only CPU-based inference was studied, and they found that the time spent in performing DNN inference amounts to only 60%, while 35.9% of the latency time is spent in the Kafka broker. We improve upon this work through two alternative implementations: An in-memory message broker (Redis), and by investigating the limitations of a fused implementation in Section 4.7.

4 RESULTS & DISCUSSION

Our goal is to better understand the overheads of DNN serving. This section presents our results in multiple important settings, spanning across different computer vision DNNs, different hardware setups, different server configurations, and the use of different message brokers—a setting in which we demonstrate considerable improvements compared to recent work [17]. In all experiments, we use throughput-optimized configurations with TrIS+TensorRT to model production servers as we argued in Section 2.3. In all cases, DNN inference is performed on the GPU, but we explore both CPU and GPU preprocessing throughout our experiments. In the following experiments, the preprocessing pipeline consists of JPEG decoding followed by image resizing and normalization.

4.1 Broad Analysis of Computer Vision DNNs

To begin our analysis of server overheads, we profile a large number of computer vision DNNs from HuggingFace in Fig. 4. A natural consequence of increasing DNN FLOPs, is that throughput decreases as shown in Fig. 4 (top). By benchmarking these models with both CPU and GPU preprocessing, we quantify the improvement from GPU preprocessing to range between -2.9% to 104% with an average of 34% across our models. As the inference FLOPs increase, the percentage of time taken in inference tends to increase as well. Fig. 4 (bottom) quantifies the average time spent on DNN inference from the point at which an image enters the host CPU, until the DNN result is returned to the host CPU. The remainder of the time is spent on preprocessing, queueing, data transfer, and postprocessing. Fig. 4 shows that these DNN overheads dominate inference requests for most models smaller than 5 GFLOPs—efficient image classification DNNs such as ResNet-50 are dominated by non-inference time. Even for models larger than 10 GFLOPs, 16–49% of the latency goes towards non-DNN functions, highlighting the importance of further analyzing these overheads.

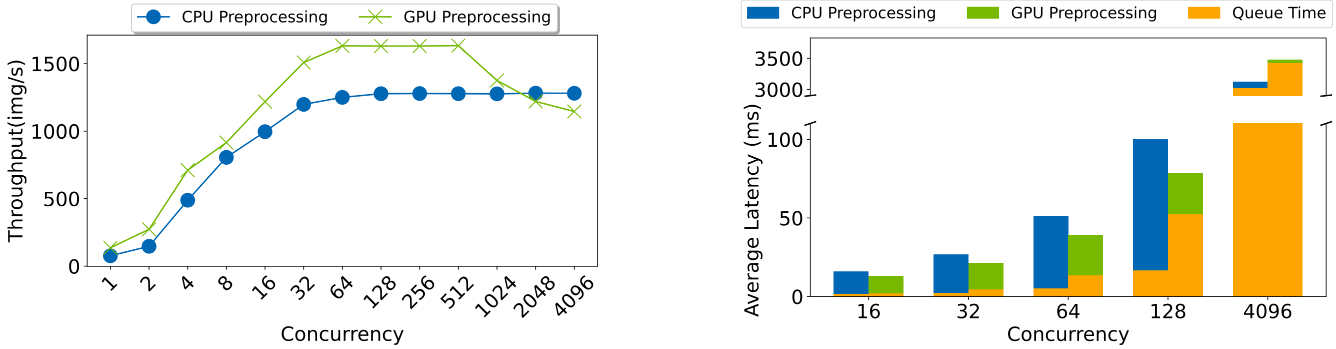

Figure 5: Throughput, average latency, and queuing time of a throughput-optimized inference server at different concurrencies.

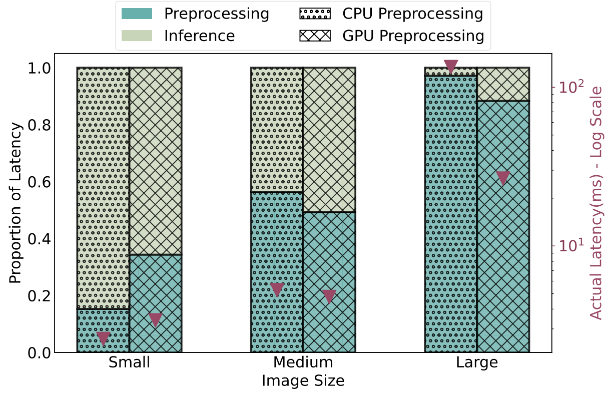

Figure 6: Latency breakdown of ViT with JPEG preprocessing under zero-load running on TrIS for different image sizes on CPU and GPU preprocessing.

4.2 Preprocessing Overhead

Fig 6 plots the latency and its breakdown into preprocessing and DNN (ViT [13]) inference under *zero-load* conditions. We plot latency for both CPU and GPU preprocessing for three representative image sizes[3] from the ImageNet dataset. Larger images consume more time in the preprocessing stage as decompression and resizing become more expensive, however, DNN inference is always performed on a standard 224×224 image. This mimics a realistic server scenario that accepts images from many clients and different resolutions/sizes, but needs to resize them all to a standard size accepted by the DNN. Interestingly, CPU preprocessing outperforms GPU in terms of latency for small images. This is likely because the GPU is vastly underutilized in this case. However, as the image size increases, GPU latency shows marked improvement, becoming significantly better for very large image sizes. A clear trend is that the portion of time spent on preprocessing increases with the image size, reaching up to **56%, 49%** in the medium image and up to **97%, 88%** in the large image in cases of CPU and GPU preprocessing respectively. This analysis demonstrates the importance of explicitly considering the preprocessing functions when designing the hardware for a datacenter server. Even with accelerated GPU preprocessing, and image sizes from within ImageNet, preprocessing dominates overall latency.

4.3 Queuing and Concurrency

Servers are commonly subjected to high request loads with an objective to maximize throughput while maintaining an acceptable tail latency. To build an optimized server, we assume that each node is running at capacity (i.e: receiving a specific number of concurrent requests), and additional requests are routed to other server nodes. Our goal is thus to maximize the throughput of each node to subsequently minimize the number of nodes required for the whole system. To assess performance scaling with concurrency in one node of a throughput-optimized system, we test our node under different concurrencies, and we record throughput and average latency. The results are illustrated in Fig 5. As concurrency increases, throughput increases but latency increases as well. GPU preprocessing generally provides higher throughput and lower latency than CPU preprocessing. However, GPU preprocessing exhibits a performance decline at very high concurrency, whereas CPU preprocessing saturates, maintaining its output rates under high load. We postulate that the decline in GPU preprocessing performance at higher concurrency levels stems from GPU memory capacity limitations. As the GPU memory saturates, preprocessed inputs queued for inference get temporarily ousted from the GPU memory, necessitating a subsequent reload—a process that incurs additional latency. Conversely, CPU preprocessing benefits from a larger main memory that can buffer images until they can be consumed by the GPU. Critically, Fig. 5 (right) shows that queuing consumes an increasing portion of round-trip latency as concurrency increases, and up to 3 seconds at 4096 concurrency. However, the optimal concurrencies in this case fall between 64 and 512 where queuing accounts for 34-91% of the latency. Even though GPU preprocessing enables higher throughput and lower latencies, more time is spent queuing in the GPU preprocessing case due to resource contention.

4.4 Throughput Bottlenecks

To understand the impact of preprocessing on throughput, we measure the throughput of GPU preprocessing and inference individually in Fig. 7. In many cases, the end-to-end throughput is aligned with either the preprocessing or inference stage, indicating the presence of a performance bottleneck in one or the other. For ViT-base, a larger model, the inference stage is often the performance bottleneck. However, with larger images, preprocessing emerges as the limiting factor, where the throughput of the end-to-end system is just 19.5% of what's achievable with ViT inference alone. The same trend is observed in the smaller ResNet-50 and TinyViT models, confirming the tangible effect that image preprocessing can have on DNN serving throughput. For medium-sized images, both inference and preprocessing can individually achieve similar throughput, indicating that both need to be optimized to be

[3] Small: 4kB 60×70, Medium: 121kB 500×375, Large: 9528kB 3564×2880.

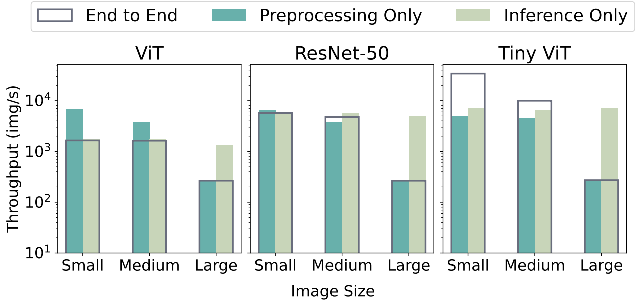

Figure 7: Comparative throughput analysis of an end-to-end inference server under different models and image sizes with

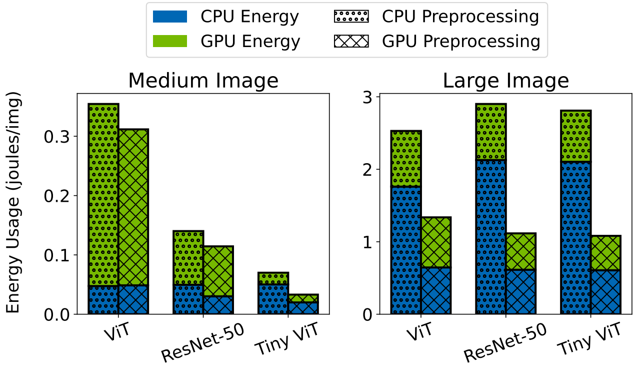

Figure 8: Energy, measured in joules, expended by the CPU and the GPU per image processed. For each model, left bar is CPU preprocessing and right bar is GPU preprocessing.

able to improve end-to-end server throughput. This highlights the importance of considering all parts of the processing pipeline to achieve some speedup. With the increase in performance of current deep learning accelerators, and GPUs in particular, it is clear that there are diminishing returns from simply optimizing deep learning performance on these systems. Instead, more holistic optimization is now needed, to include additionally heavy preprocessing tasks, especially for computer vision.

An outlier in our results occurs in the case of small/medium images and TinyViT: end-to-end system performance is *faster* than inference-only performance. We root-caused this issue to data transfer overheads. Particularly, we only transfer the compressed image to the GPU in the case of end-to-end preprocessing+inference. However, in the inference-only case, we transfer the decoded raw image which is ~5× larger. Even for very fast preprocessing and inference, data transfer may start to limit overall system performance. Many DNN users are beginning to use such optimized models server-side to save on costs, and our results highlight potential throughput bottlenecks in these cases.

4.5 Energy Utilization

Fig 8 plots the energy utilization per image in different scenarios. In general, CPU-based preprocessing results in higher energy usage across the board. This is likely because of the lower device utilization and increased data transfers and memory accesses when the CPU is used for preprocessing and the GPU is used for inference. When moving from the medium image to the large image, we see a clear increase in CPU energy utilization. The reason is obvious in the case of CPU preprocessing: a larger image requires more computing time and energy. For the case of GPU preprocessing, more CPU memory accesses and PCIe transfers are needed to send the larger image to the GPU, resulting in increased energy consumption. Comparing the GPU portion of energy utilization, it is consistently smaller when the GPU is doing both preprocessing and inference even though it is doing more work in this case. However, the improved device utilization over-compensates for the additional work, resulting in an overall decrease in average energy per image.

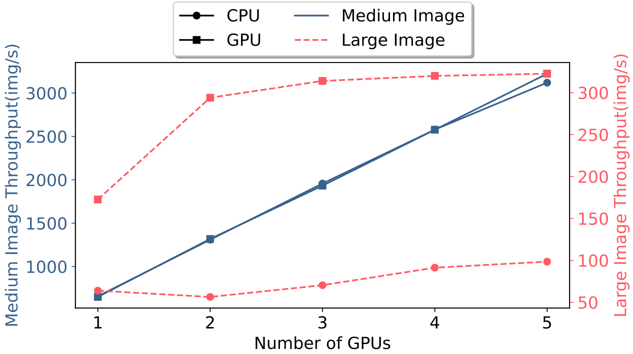

Figure 9: Throughput variation as we increase the number of GPUs for the medium and large images on different hardware used for preprocessing.

4.6 Multi-GPU Scaling

To scale performance, it is common to include multiple accelerators connected to each host CPU in a server node. We study the scaling of ViT-base inference with multiple GPUs in Fig. 9. Throughput for our medium image exhibits a linear scaling with more GPUs—this happens for both CPU and GPU preprocessing. However, for larger image size, where preprocessing is the performance bottleneck, the increase in GPU count doesn't always translate to an increase in throughput. In case of GPU preprocessing, transitioning from a single GPU to dual GPUs introduces a notable throughput enhancement. However, further GPU additions result in marginal gains, exposing an underlying performance bottleneck in preprocessing. Running inference only shows linear scaling pattern which confirms that inference is not the bottleneck. When using the CPU for preprocessing, there is minimal change in performance as we increase the number of GPUs since performing the preprocessing consumes a majority of the time and CPU processing cycles, therefore the additional GPUs are wasted, waiting for incoming inference requests from the CPU.

4.7 Message Brokers in Multi-DNN Systems

In this section, we analyze a system that contains multiple DNNs connected to each other via a broker similar to the system analyzed in [17], illustrated in Fig 10. A message broker is useful when two connected processes produce and consume outputs at different rates. This is the case for the face-detection-then-identification pipeline that we are investigating because one frame can contain multiple faces detected in the first stage using Faster R-CNN, followed by multiple invocations of a face identification DNN in the second stage using FaceNet. We analyze this pipeline under multiple configurations: Using Apache Kafka as described in prior work [17], using an in-memory message broker called Redis, and fusing the components of the system into a single process without a message

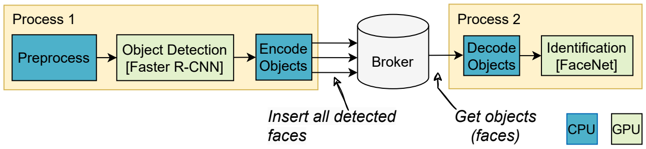

Figure 10: A multi-DNN system for face identification.

broker in between. Fig 11 shows performance improvement with a memory-backed Redis broker, compared to the disk-backed Apache Kafka, leading to a 125% improvement in overall system throughput and 67% improvement in zero-load latency at 25 faces per frame with Kafka taking 71% and Redis taking 6% of the total latency. It is also observed that the Fused system performs best when the number of detected faces is small. In this case, the inefficiency of running two stages with different rates is outweighed by the overheads of either message broker. However, with a high number of detected faces, the gap in throughput between the Fused system and the Redis broker decreases, and eventually Redis outperforms Fused when 9 or more faces are detected. Our results highlight two key observations. First, that message brokers may not always be needed in a multi-DNN system—it depends on the rate mismatch and workload of each DNN. Second, that in-memory brokers such as Redis significantly outperform disk-based message brokers for multi-DNN systems, thus revising the reported overhead of a face detection system down to just 6% instead of 36% [17].

5 CONCLUSION

In this paper, we benchmarked vision DNNs on a throughput-optimized inference server to analyze system performance, identify performance bottlenecks, and quantify DNN serving overheads. Our broad analysis of vision DNNs clearly demonstrated that inference does not dominate performance on modern GPUs, especially for DNNs less than 5 GFLOPs. We proceeded investigate the sources of performance bottlenecks and found that standard preprocessing on common image sizes can account for a large portion (>50%) of the zero-load DNN serving latency, even when accelerated GPU preprocessing is used. Under high concurrency, we further found that queuing accounted for ~60% of total latency. From a throughput perspective, we found that modern GPUs have become very efficient in processing DNNs and once again preprocessing could limit system performance, especially for single CPU, multi-GPU systems. Accelerating preprocessing on the GPU using the NVIDIA DALI library can alleviate these scaling limitations but only to a certain extent due to batched preprocessing. However, overall performance can still be throttled by preprocessing beyond two GPUs. Finally, we investigated the impact of message brokers between two DNNs and found that prior work has overestimated their overhead because of the reliance on Apache Kafka. We additionally investigated Redis and a Fused implementation, showing that the broker overhead can be as low as 6% and we boosted performance by 2.25× compared to prior work. Our work provides a clearer understanding of DNN servers for computer vision tasks, and lays the foundations for optimized system design.

ACKNOWLEDGMENTS

This project is supported in part by Intel Corporation funding. We would like to thank Mahesh Iyer, Nilesh Jain, Aravind Dasu, Gregg Baeckler, and Ilya Ganusov for insightful discussions and feedback.

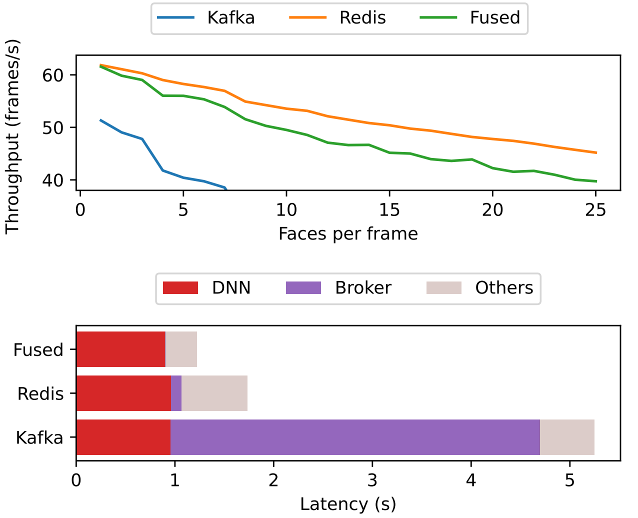

Figure 11: Throughput and latency breakdown of a multi-DNN systems with different brokers.

REFERENCES

[1] 2023. https://blog.youtube/news-and-events/using-technology-more-consistently-apply-age-restrictions/

[2] 2023. https://developer.nvidia.com/blog/leveraging-hardware-jpeg-decoder-and-nvjpeg-on-a100/

[3] 2023. AI Matrix. https://aimatrix.ai/en-us

[4] 2023. Business Insider: Facebook Users Are Uploading 350 Million New Photos Each Day. https://www.businessinsider.com/facebook-350-million-photos-each-day-2013-9

[5] 2023. ChatGPT sets record for fastest-growing user base - analyst note. https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/

[6] 2023. NVIDIA Data Loading Library (DALI). https://developer.nvidia.com/dali

[7] Mohamed S Abdelfattah et al. 2018. DLA: Compiler and FPGA overlay for neural network inference acceleration. In *2018 28th international conference on field programmable logic and applications (FPL)*. IEEE, 411–4117.

[8] Robert Adolf et al. 2016. Fathom: Reference workloads for modern deep learning methods. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–10.

[9] Gene M Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*. 483–485.

[10] Wesley Brewer et al. 2020. iBench: a distributed inference simulation and benchmark suite. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–6.

[11] Wesley Brewer et al. 2020. Inference benchmarking on HPC systems. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–9.

[12] Cody Coleman et al. 2017. Dawnbench: An end-to-end deep learning benchmark and competition. *Training* 100, 101 (2017), 102.

[13] Alexey Dosovitskiy et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[14] Amin Firoozshahian et al. 2023. MTIA: First Generation Silicon Targeting Meta's Recommendation Systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.

[15] Norman P. Jouppi et al. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. arXiv:2304.01433 [cs.AR]

[16] Vijay Janapa Reddi et al. 2020. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 446–459.

[17] Daniel Richins et al. 2021. AI tax: The hidden cost of AI data center applications. *ACM Transactions on Computer Systems (TOCS)* 37, 1-4 (2021), 1–32.

[18] Huaizheng Zhang et al. 2020. Inferbench: Understanding deep learning inference serving with an automatic benchmarking system. *arXiv preprint arXiv:2011.02327* (2020).

[19] Hongbin Zheng et al. 2020. Optimizing memory-access patterns for deep learning accelerators. *arXiv preprint arXiv:2002.12798* (2020).