Optimizing Inference on Heterogeneous Platforms

Param Damle
psd9vgc@virginia.edu
University of Virginia
Charlottesville, Virginia, USA

Pawan Jayakumar
pj8wfq@virginia.edu
University of Virginia
Charlottesville, Virginia, USA

Figure 1: (From left to right): Nvidia RTX 4070 Super GPU, Coral Edge TPU USB Accelerator, Intel i7 14700K CPU.

ABSTRACT

Machine learning inference is becoming an increasingly popular workload in modern data centers due to the explosion of interest in large language models. This paper explores optimizing inference on heterogeneous platforms composed of multiple different types of accelerated processors. More specifically, we run experiments to investigate whether the inclusion of Google's Edge Tensor Processing Unit (Edge TPU) along with an Nvidia Graphics Processing Unit (GPU) can speed up inference on deep neural networks. We explore data level parallelism which involves running the same workload across both the TPU and GPU, as well as model level parallelism where the model is split into separate portions, each running on a different device. Through our research, we find that while the TPU is optimized for small-scale, simple neural network workloads, it does not scale well and delegating large batch sizes or more intricate operations to it would result in suboptimal performance.

KEYWORDS

Machine learning, neural networks, hardware accelerators, Edge TPU, GPU, inference, heterogeneous platforms

1 INTRODUCTION

Machine learning (ML) inference has become a significant workload in data centers, partly due to the popularity of services such as OpenAI's Chat-GPT, which has become the fastest service to reach 100 million monthly users [5].

As more accelerators become present in consumer devices, an interesting exploration resides in whether heterogeneous computing (concurrently leveraging multiple different accelerators for the same task) can outperform using only one processor. The motivation is that rather than sitting idle, we can use better software to expand the capabilities of an accelerator and apply it to more applications. This would save energy in both data centers and mobile computing platforms. Further, user devices such as laptops, microcontrollers, and Internet of Things (IoT) devices are embedded with a variety of processors that present the potential for tailored inference optimization. Our project specifically tackles optimizing ML inference on commercial laptops that have onboard graphics processing units. We were able to connect an Edge **tensor-processing unit** (TPU) via USB, transforming the laptop into a heterogeneous platform when augmented with the in-built CPU and GPU.

1.1 Background

Google's Edge TPU [9] operates as a compact version of its regular datacenter TPU [6] with a smaller systolic array structure to

optimize matrix multiplication and accumulation, the operations that are crucial to linear algebra and its applications (digital signal processing, neural networks, etc). The Edge TPU in particular is designed as a response to greater inference demand on personal devices (e.g. mobile phones, smart watches) and IoT devices like smart cameras. Google (via coral.ai) has published documentation [8] on how to run deep neural networks on the Edge TPU using the TensorFlow Lite (TFLite) library. This high-level compilation tool allows us to define models in Keras, a high level framework in Python. TFLite then quantizes, optimizes, and compiles the model into a form that can run on the Edge TPU.

The Edge TPU has significantly smaller energy footprint compared to a laptop GPU like the Nvidia RTX 2060 Mobile. It has also been designed for latency minimization, rather than the throughput maximizing design of cloud-scale architectures. This means that while it is fast, there may be data bandwidth limitations, especially when the mode of connection is USB 2.0 as with the accelerator in this study. We are interested in the possibility that by combining accelerators with different strengths, this setup can provide better performance and flexibility to adapt to various types of workloads.

1.2 Prior Work

Prior work by Hsu and Tseng [3] devised Simultaneous and Heterogeneous Multi-threading (SHMT) [4], a protocol for scheduling work on a CPU that's connected to a GPU and multiple Edge TPUs. Their experiments showed an impressive 1.95× speed up with a 51% reduction in energy usage when using a heterogeneous set up as opposed to their baseline GPU set up. Their method is a form of the more general data parallelism paradigm where the same model is placed on multiple devices, with input data being split amongst them. While they focused on a variety of filters and operations relevant to image and signal processing, we plan to study specifically ML **inference** (the act of running a pre-trained model on newly received input samples to generate new outputs and predictions), as this is the intended purpose of the Edge TPU and conceivably the most frequent use case of edge devices.

Other work [1] has described splitting up parts of a convolutional neural network layer-wise to run across devices in an edge computing cluster. The motivation for this work was that the Edge TPU can help accelerate fine tuning models on the edge, with the majority of the workload running on more performant processors. While this aspect of the work does not have any relevance, we find their method of model parallelism (sharding the model across multiple devices) to be an avenue worth exploring.

2 METHODS

2.1 Inference Benchmarking

Before starting our experiments, we wanted a better picture of the performance of our Edge TPU, and how that compared with both a GPU as well as a full fledged cloud TPU in a Google datacenter. While the Edge TPU is inherently going to be weaker because it was designed for power-constrained edge devices, it is unfortunately the only form factor of the TPU that is commercially available.

We benchmarked the Edge TPU on two tasks: simple matrix vector multiplication and a series of inference tasks on progressively larger fully connected neural networks. The motivation behind these tasks is twofold:

(1) the TPUs design was centered around matrix multiplication (systolic array-based multipliers form the core of the chip)
(2) fully connected neural networks are the simplest representative workload that the TPU does quite well on.

For matrix multiplication, we created matrices of sizes varying powers of two from 64 to 1024. Since the TFLite compiler for the Edge TPU was only designed to handle ML workloads, we had to design a single-layer neural network with no activation function to encode a matrix multiplication task into the requisite model type.

For the neural network, we started with a base model that had five fully connected layers and progressively scaled the number of neurons in each layer. The number of neurons in each layer is $int(2^{\lambda * \nu})$ where λ is the scaling factor in {1,2,...,7} and ν is the layer scale for each layer in the network, $\nu \in (1.5, 1.3, 1.1, 1)$. The number of neurons in the last layer is simply 2λ.

The full results of both these benchmarks, found in Section 3, required us to assume a common scaling for the TPU and GPU. Thus, we needed to either multiply the performance of the Edge TPU to be on par with the laptop-/server-grade GPU or downgrade the GPU performance to be on par with edge devices to allow for more interesting comparison. We know from [2] that the latest generation cloud TPUs (TPU V5e) from google achieve 393 FLOPs. This is about 100x the performance of the Edge TPU. We believe this multiplier can be generalized to our experiments because the FLOP metric for both devices was computed by assessing it over a suite of different models.

2.2 Data Parallelization

Data parallelization involves running the same model across multiple devices and aggregating the results afterward. For *inference*, this is theoretically quite easy to set up, as aggregating results simply involves transferring data to a centralized location. *Training* a model with this method introduces the challenge of needing to combine losses, gradients and applying those changes to all models in an efficient manner.

What is simple in theory often presents practical obstacles. PyCoral (the framework used to run inference on the Edge TPU) and vanilla TensorFlow are not both supported in the same Python file as they clash on importing an Interpreter Wrapper. We realized we could not call interpreters for both the GPU workload (which relied on a vanilla TensorFlow interpreter) in the same Python file as the TPU workload (which relied on the PyCoral-specific interpreter) because import of any Python module runs the whole module automatically, inevitably producing the conflicting definition error. We averted this by launching separate threads for separate devices using the Python `subprocess` library to call entirely self-contained Python scripts from each thread.

2.3 Layer Parallelization

Another method for optimizing inference involves sharding the model layer-wise into two parts and running one on the GPU and the other on the TPU.

When using GPU for inference, it takes time for data to be transferred from host to device. This cost can be amortized by exploiting

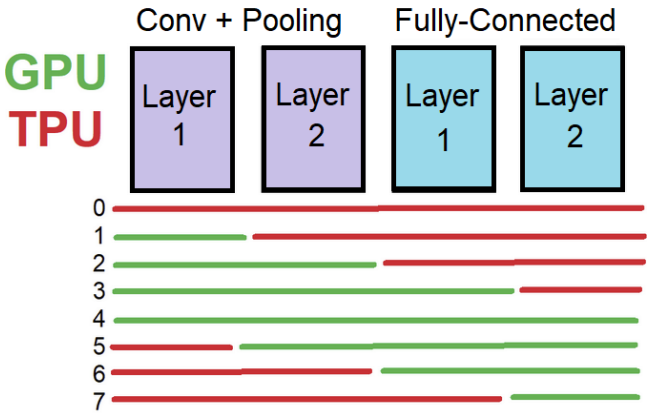

Figure 2: Layer split configurations depicting what layer blocks of a CNN are run on each device

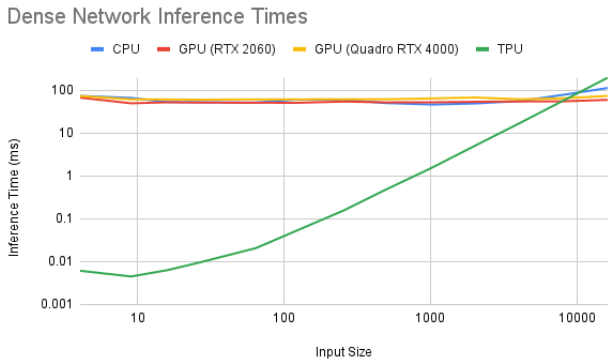

Figure 3: Comparison of per-device runtimes for simple neural network inference.

SIMT architecture and load more data all at once, and then performing batched inference (which also allows for weight reuse). As GPUs are throughput-oriented, batching drastically improves performance. For example, processing a singular input through our CNN model took about half a second while using a batch size of 1024 dropped the per instance inference time to just 4.8 ms.

We defined a convolution neural network (CNN) with four blocks of layers each. We created two types of blocks: convolution and dense (fully-connected). The Convolution block consisted of two 2D convolution layers separated by a depth wise 2D convolution layer, which are followed up by a Max Pool layer. The dense block contains two fully connected layers separated by a dropout layer. The first CNN we created has three convolution blocks followed by a dense block while the other has two convolution blocks followed by two dense blocks. The last convolution block in both models uses global max pooling to reduce the the tensor rank from four to one.

Our model parallelism experiment consisted of separating the CNN at a block level. We defined 8 different model split configurations which placed a different number of consecutive blocks on the Edge TPU and GPU respectively. This can be seen in Figure 2: configuration 0 was running everything on the GPU. Configurations 1-3 allowed the GPU to run the first sequence of blocks. Configurations 4-7 are symmetric with the difference being now it's the TPU running the first sequence of blocks. Since the layers switched from entirely convolutional to entirely fully connected, and we assumed that data transfer would present overheads, we did not consider schemes where a non-contiguous subset of layers was delegated to a given device.

3 RESULTS

3.1 Benchmark results

To start, the matrix multiplication results in Table 1 demonstrate that the Edge TPU is only performant at matrix sizes up to 128×128, and both the TPU and GPU are beaten out by the CPU until the matrices scale beyond size 1024×1024. This indicates that when divvying up a complex ML workload, delegating a simple matrix multiplication task to the Edge TPU will likely be suboptimal unless the other processors are busy with more computationally demanding work.

However, this does not imply that the Edge TPU is inefficient at tasks that largely involve matrix multiplication. For example, a simple feed-forward neural network involves layers of matrix multiplication with activation functions like ReLU in between. When applied to the model described in Section 2, Figure 3 depicts our observation of TPU's strengths and weaknesses. On one hand, the Edge TPU scales poorly (linear on a log-log plot) while the CPU and GPUs tested virtually stayed constant. On the other hand, the objective inference time of the Edge TPU was orders of magnitude faster than that of the more powerful devices for inputs less than 10^4 features (roughly representing a 100×100 black and white image). If any relevant edge devices are able to scale down the input image (or otherwise reduce the dimensionality of input data of another format) to common input sizes, the Edge TPU still demonstrates a speedup in inference of a simple neural network.

3.2 Data Parallelization

After covering the runtime boost of entirely executing neural inference on a processor of choice, we now examine the potential for heterogeneous multithreading, where different aspects of a model (either the data or layers) run on distinct processors connected to a host. Figure 5 in the Appendix shows the runtime impacts of delegating a different proportion of the inference batch to each device (TPU, GPU, CPU) on a heterogeneous system for a dense neural network. The comparison of the plots across the first row demonstrates the inference times of each device alone depending on the allocation scheme; as expected, the TPU's isolated inference times scales exactly proportionally to the batch size since it processes each sample serially, while the GPU and TPU runtimes have an inconsistent effect due to their more generalized and complicated programming. These plots represent the timing of just the inference invoking command on each device, excluding all tensor loading and retrieval, data transfer, and multithreading overhead. Adding up this runtime across the three devices yields the "theoretical" total runtime plot (bottom left), which assumes all overheads are 0. As

Matrix Side Length	64	128	256	512	1024
Edge TPU (with memory loading)	**0.1518**	**0.3469**	0.9515	3.5786	14.0721
NVIDIA GTX 1650 (with memory loading)	0.2009	0.3865	**0.4294**	**0.6712**	**1.3720**
Edge TPU (inference only)	0.0586	0.2931	0.9932	3.4194	14.0038
NVIDIA GTX 1650 (inference only)	0.0624	0.1252	0.0718	0.0930	**0.1106**
Intel Xeon @ 2.2GHz	**0.0412**	**0.0687**	**0.0518**	**0.0661**	0.1555

Table 1. Table of matrix multiplication time (in milliseconds). All results shown are the average of 100 trials.

mentioned in the previous section, the TPU is significantly faster at these smaller-scale simple neural workloads, so the allocation scheme without GPU involvement, specifically exclusively running all 10 of 10 samples on the TPU, yields the lowest theoretical runtime. However, upon examining the actual runtime plot (which measures the total time from when the separated device threads are launched to when they are joined), we observe the maximum runtime occurring consistently when the TPU is invoked at all, leading to large ratios between the actual and theoretical runtimes. After examining the live execution of inference on the Edge TPU, it was determined that the overheads of transferring data over USB 2.0, which connects the accelerator to the host device, contributes an approximate 15 second minimum execution time, regardless of the actual inference time. Additionally, when testing the system, we had to implement an extra 3 second delay between inferences on the Edge TPU. Without this, either the tensors were not effectively cleared from the USB Accelerator or it was otherwise not completely reset to a starting state, and immediately calling another inferencing task yielded an error about missing the Edge TPU runtime library.

To analyze the runtime of the Edge TPU without USB overhead, we attempted to test a larger, more complicated model with larger sample sizes, to run a workload inherently longer than 15 seconds on the Edge TPU. Using the AlexNet convolutional neural network architecture [7], we ran the same experiment and split up to 20 samples for concurrent inference across our 3 devices. Figure 6 in the Appendix shows that while the Edge TPU demonstrated similar linear scaling and the other two processors had sporadic performance in isolation, the actual and theoretical runtimes aligned much more closely this time, staying within an order of magnitude on average. We find that when the TPU requires a large amount of time to inference (here due to the complex nature of a CNN), the USB overhead disappears and the TPU runtime vastly dominates the overall runtime. A consequence of this is that the actual runtime more closely aligns with the theoretical sum as the TPU takes on a greater load, likely because any overheads of the CPU, GPU, or USB interface are amortized over the large amount of time required by the TPU alone.

This means that although the TPU is not the best at scaling to harder and larger workloads, the fault was within the data interface and not the processor itself. Further, as the ratio plot (bottom right) in Figure 6 shows, the TPU is the most reliable of the tested processors in terms of aligning actual runtime with inferencing overhead with the theoretical runtime. It was overall surprising how proficient the CPU was at these neural network workloads, but this could be due to easier access of host resources and threading, generalizability of its processor, or a lack of optimization from cuDNN and TensorFlow.

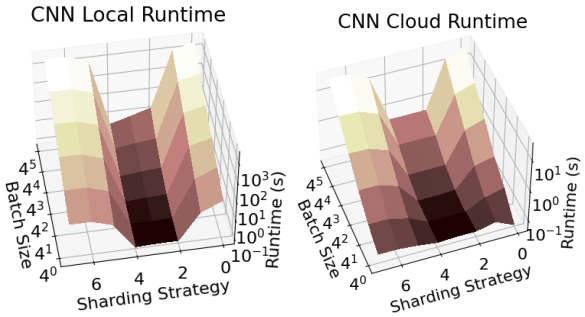

Figure 4: Inference time for various layer parallelism schemes.

3.3 Layer Parallelization

As shown in Figure 4, our various layer sharding strategies from Figure 2 universally resulted in inference times that scaled with the amount of samples that were processed on that device. Specifically, we see the TPU's serial limitation causes strategies that heavily delegate to the TPU (e.g. 0, 7) scale much worse than those that delegate to the GPU (e.g. 4).

To better compare the true performance of a heterogeneous system, we apply the scaling mentioned previously and scale up the Edge TPU's performance by 100 to represent a Cloud TPU, the overall runtime becomes much more competitive. While the runtime still scales in the GPU's favor, the minimum runtime for just single-sample inference (which edge devices are largely designed for) is actually provided by the purely-TPU sharding strategy.

Another noticeable feature of these plots is that strategies 2, 3, and 4 have comparable performance through multiple batch sizes. In all of these strategies, the GPU handles all convolutional layers and the remaining fully connected layers are distributed between the GPU and the TPU. This means that when the TPU is running simpler workloads (e.g. the basic matrix multiplication required in regular neural networks), it operates with comparable performance to the GPU.

4 CONCLUSION

As privacy, energy, and connectivity concerns drive computation from centralized cloud centers to edge processing devices, and a push for operation-specific accelerators enables heterogeneously composed processor layouts, the ML Acceleration community will need to contend with optimizing workloads for a specific blend of processing units. This work establishes that models perform best when generalizable compute (e.g. custom ML filters and layers) is

delegated to CPU, complex matrix-based operations (e.g. convolutional filters and pooling layers) are delegated to GPU, and simple, small scale matrix-based fully-connected layers are delegated to TPU. While the Edge TPU is shown to not scale adequately to batch or workload size, its raw performance on small-scale inferencing workloads is significantly better than that of other processors, not considering data transfer overhead.

4.1 Limitations

Several software and hardware limitations prevented us from our ideal comparisons. For example, TFLite delegates for GPU (i.e. TFLite-compatible execution on GPU) are only supported for IOS and Android. This means we could not benchmark the *exact* same workflow on the TPU as the GPU. As TFLite models are created from a Keras model, we used this Keras model to run on the GPU. However, since the Edge TPU still had to run a TFlite model, we could not use any operations incompatible with TFLite compilation, including those attributed to larger-dimension tensors and newer approaches like transformers.

As it was optimized for edge compute, the Edge TPU has an extremely small form factor, thus producing an inordinate amount of energy per mm^2 and only being able to handle smaller-scale problems. Our experimentation showed that even operations compatible for execution on the TPU, such as convolutional layers, vastly underperformed the GPU compared to simpler matrix multiplication-based workloads that better aligned with the systolic array architecture.

Another limitation the Edge TPU faced was that it was constrained to only 1 inference at a time (no batch processing) as well as only being able to support UINT8 quantization. These two limitations reduced the exploration space of inference optimizations because we could not tweak these parameters to improve performance. The Edge TPU being constrained to serial inputs may be apt for *most* edge workloads (e.g. a camera system processing one frame at a time, or a user's phone translating one of their recordings at a time), we still see it as a massive limitation that detracts from the feasibility of using it to accelerate machine learning in comparison to a GPU or CPU.

A final major limitation is that our system setups were not configurable either. The way data flows between all these devices is fixed by the laptop manufacturer. We did not have access to many different configurations of laptop GPUs and interfaces for the TPU. For example, a desktop has PCIe slots and Coral produces a version of the TPU that can be placed in that slot. GPU performance is also drastically better on a desktop because of better cooling capabilities and less energy constraints.

4.2 Future Work

This work does not include a comparison with the end-to-end runtime of running inference at a datacenter ("the cloud") and communicating with an edge device to retrieve inputs and send results. Thus, the runtimes presented are just in comparison to the other devices in the study, assumed to all be connected to a host edge device. Other work could develop a flexible model that sends data to the cloud for batched, centralized execution when connectivity is high and local execution when connectivity is low to minimize the performance hit of running inference on the edge.

As we mentioned earlier, future studies involving the Edge TPU would benefit from studying heterogeneous multithreading with a PCIe-connected TPU to overcome any overheads presented by USB 2.0. Further, software releases that allow for Edge TPU runtime execution on Windows Subsystem for Linux and a TFLite delegate for Linux GPUs would speed up the research and development workflow.

Lastly, future work could pursue the research directions this study was unable to feasibly pursue. Primarily, this would entail establishing an accuracy-runtime tradeoff of training and inference depending on distinct or heterogeneous quantization schemes that optimize a model to a given processor, input scaling that optimizes performance for edge devices, or interconnects that tailor datapaths between processors to represent connections between layers in a sharded model.

ACKNOWLEDGMENTS

We would like to acknowledge Professor Kevin Skadron for helping us better align our research directions to the project constraints, providing the Edge TPU Accelerator fundamental to this study, and for his continual mentorship throughout this project.

REFERENCES

[1] Erik Dubois. 2021. Shared Learning Among Distributed Edge Devices Using Coral Edge TPU Machine Learning Engines. https://apps.dtic.mil/sti/trecms/pdf/AD1150924.pdf

[2] Google. [n. d.]. how Cloud TPU V5e Accelerates Large Scale AI Inference. https://cloud.google.com/blog/products/compute/how-cloud-tpu-v5e-accelerates-large-scale-ai-inference. Accessed: 2024-03-16.

[3] Kuan-Chieh Hsu and Hung-Wei Tseng. 2021. Accelerating applications using edge tensor processing units. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri, USA) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 56, 14 pages. https://doi.org/10.1145/3458817.3476177

[4] Kuan-Chieh Hsu and Hung-Wei Tseng. 2023. Simultaneous and Heterogenous Multithreading. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (Toronto, Canada) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 137–152. https://doi.org/10.1145/3613424.3614285

[5] Krystal Hu. [n. d.]. 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/. Accessed: 2024-03-16.

[6] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. *SIGARCH Comput. Archit. News* 45, 2 (jun 2017), 1–12. https://doi.org/10.1145/3140659.3080246

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (may 2017), 84–90. https://doi.org/10.1145/3065386

[8] Zhekai Li, Zhen Dong, Mingfei Guo, and Lingran Zhao. [n. d.]. Edge TPU inferencing overview. https://coral.ai/docs/edgetpu/inference. Accessed: 2024-03-14.

[9] Q-engineering. [n. d.]. Google Coral Edge TPU explained in depth. https://qengineering.eu/google-corals-tpu-explained.html. Accessed: 2024-03-14.

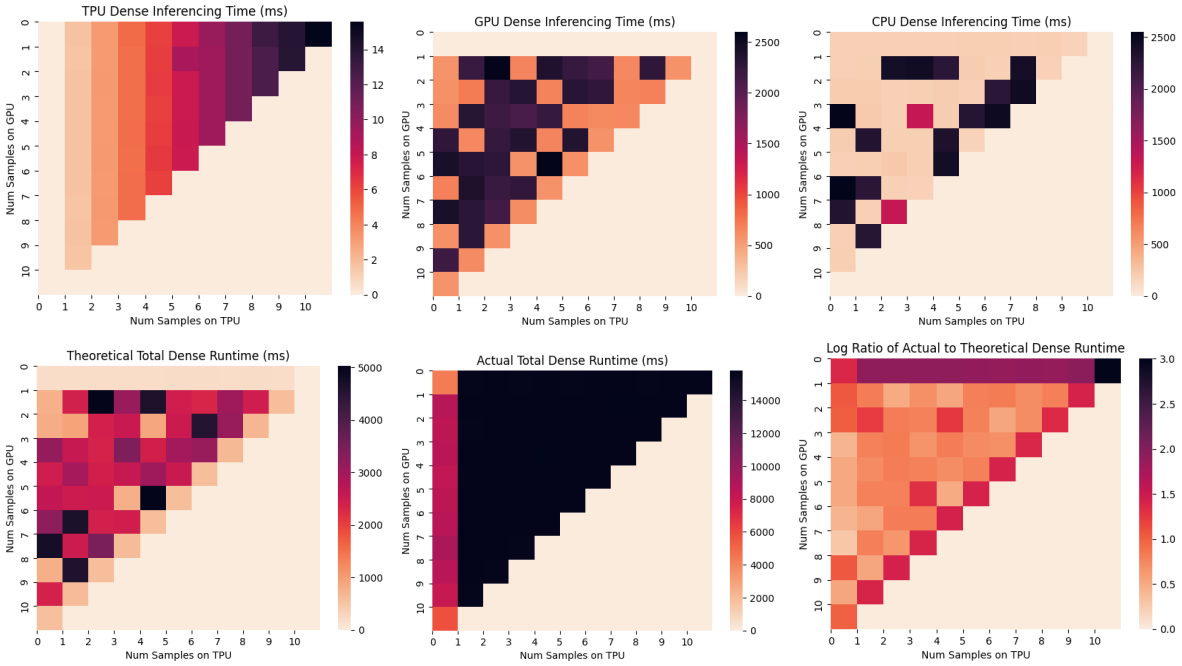

Figure 5: Data parallelism between TPU (x), GPU (y), and CPU (remainder) running a dense, fully-connected neural network.

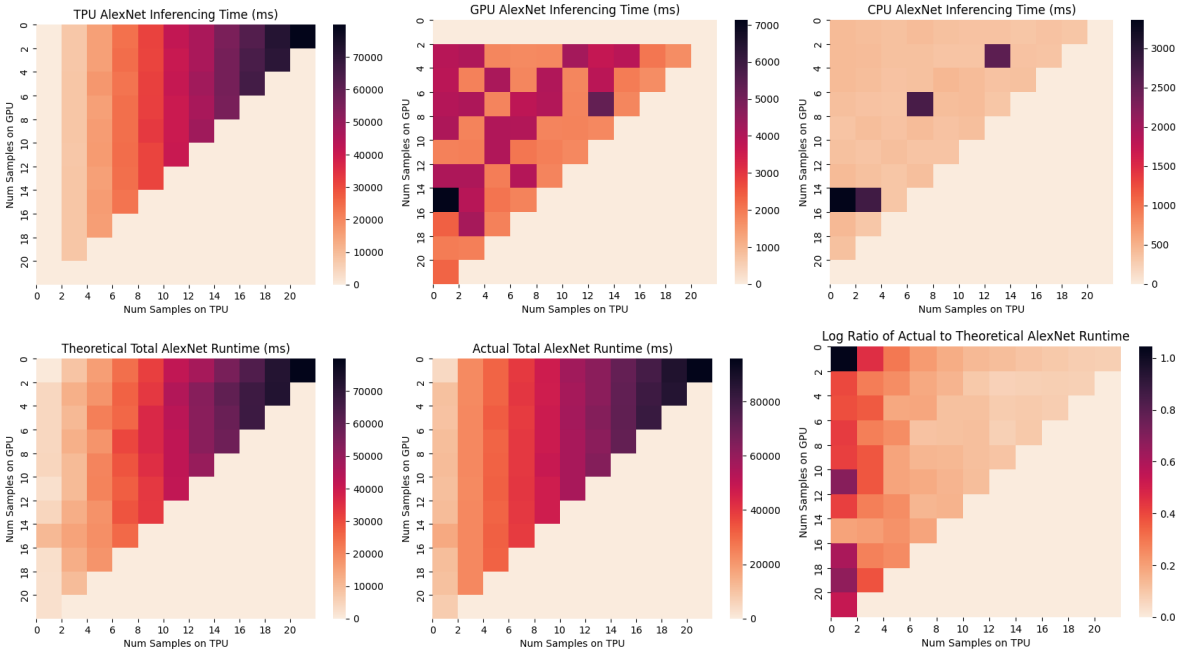

Figure 6: Data parallelism between GPU (y), TPU (x), and CPU (remainder) running the AlexNet convolutional architecture.

A DATA PARALLELISM RESULTS

Above, we provide heatmaps that show the inferencing runtime for a given partition scheme, which is simply how many samples each device (between TPU, GPU, and CPU) inferences between synchronization barriers. The dense network workload distributed 10 samples while the more computationally challenging AlexNet inference workload distributed 20 samples across the 3 processors.