MACHINE LEARNING INFERENCE FRAMEWORK BENCHMARKING INTERNSHIP REPORT

Applied Data Science Internship / Internship Project
[Data Science Course]

Company: Address: Supervisor: Position:	PERFACCT GmbH PERFACCT, Performance Acceleration Technologies, August-Bebel-Straße 27 14482 Potsdam, Germany Tobias Jaeuthe CEO & Co-Founder
Name: Email: Matrikel-Nr:	Yuvraj Dhepe dhepe@uni-potsdam.de 820418
Internship Period: Semester at the time of internship:	15.11.2023 - 30.03.2024 3

INTERNSHIP COMPANY: PERFACCT

Perfacct uses analysis and measurements to reveal optimization potential in computationally intensive simulations. The company recommends user actions to save computing time as well as hardware and energy costs.

TIMING OF THE INTERNSHIP

The internship began in the 3rd semester. The total duration of internship was from 15.11.2023 to 30.03.2024. The daily time spent was about 8 hours, with the total work hours completing 360 hrs.

AREAS OF ACTIVITY AND TASKS DURING THE INTERNSHIP

INTRODUCTION

Recent developments of large language models have led deep learning to become one of the most popular and increasingly adopted fields of artificial intelligence. Deep learning has bloomed in recent years mainly because of availability of large variety of data, advances in data-parallel-processing hardware like GPUs and due to increasing number of open-source deep learning software frameworks (Liu., et al 2018).

Deep learning involves the stage of model training, in which neural nets are trained on data, where the neural nets identify the patterns in the data. Once a model is suitable for a given use case trained well to achieve certain results, then the model is deployed in the real world, where real data is passed through the neural network. When deployed, the model must predict real-world data, based on the patterns learnt. This process of passing the real-world data to obtain predictions is known as inference.

While performing inference the computationally intensive neural networks devour the GPUs Virtual RAM (VRAM). The VRAM hunger of models could be attributed to performing mathematical computations parallelly on GPUs, large model sizes in terms of memory footprint to store the

parameters, intermediate activations during the inference and lastly complexity of the mathematical computations being carried out by the model.

It can happen in real-time that the number of GPU resources available fails to fulfil the demand of inference requests of clients. Especially in use-cases of quality control or automated driving, it can lead to huge loss on both provider and consumer side. Thus, it is very crucial for the organizations, deploying deep learning-based models to use efficient machine learning models based on their use-case. It is not only using an efficient model; however, also crucial to deploy it using the right inference framework, suitable for their hardware. ML scientists, developers and enthusiasts have proposed diverse optimization inference frameworks to accelerate the predictions for end-user applications (Pochelu, Pierrick. 2022).

Currently there is no one size fit for all purpose inference framework, in terms of performance. In this internship, we tried to test various inference frameworks and benchmark them on the metrics of throughput, latency and power efficiency delivered by the inference framework. For performing an extensive and robust comparison we even tested the inference frameworks on 3 different GPUs. Briefly the results varied across different GPUs thus confirming to the notion that each inference framework performed different optimizations specific to the hardware and leading to different performance, and it is crucial to choose a specific framework as per one's own use case and hardware. Finally, for the organization we were able to decide which inference framework will be performance-effective, and useful in real-time.

RELATED LITERATURE

Techniques such as pruning, and quantization are the major techniques discussed by recent literature for deep neural networks acceleration. One such paper explores ways to speed up neural network inference by balancing model complexity and computational demands. The authors discuss distinct types of convolutional neural networks (CNNs) and introduce pruning and quantization techniques to reduce model size and computational requirements. Various pruning and quantization methods are compared, considering their impact on accuracy and computational efficiency. Additionally, the paper introduces adaptive CNNs and reinforcement learning-based approaches to enhance inference performance automatically.

Furthermore, the paper evaluates these techniques across different frameworks like TensorFlow and Torch, highlighting their strengths and weaknesses. It emphasizes the importance of hyperparameter optimization for achieving the best performance and provides guidance on selecting optimal values for parameters such as layer types and pruning rates (Tailin, Glossner, Wang, Shi, and Zhang. 2021).

Another group of authors conducted a thorough performance evaluation of workflows aiming to accelerate Torch models using TensorRT on hardware platforms with limited resources, primarily focusing on local computation of CNN (convolutional neural networks) model inference. It assesses performance in terms of latency, throughput, and GPU memory usage, while also discussing the strengths and weaknesses of each workflow. Results suggest that ONNX to TensorRT Conversion yields the best overall performance for enhancing Torch model inference, while ONNX Runtime with TensorRT Integrated demonstrates lower latency and higher throughput compared to Torch-TensorRT. However, no single workflow emerges as universally optimal, with recommendations varying based on deployment scenarios (Zhou, Yuxiao, and Kecheng Yang. 2022).

Lastly another relevant study focused on benchmarking various optimized neural networks, via multiple inference frameworks and goes on benchmarking an ensemble of neural networks on these frameworks. The discussed post-training optimization in paper aims to refine deep learning inference frameworks after model training, leveraging high-level model representations to optimize computational graphs and code representations for specific hardware (Pochelu, Pierrick. 2022). The author further discusses high-level optimizations, such as fusing, constant-folding and low-level

optimizations which fine-tune framework performance by converting computational graphs into hardware-optimized code, incorporating techniques like sub-expression elimination, vectorization, and memory caching.

From this literature review, we found the best frameworks to use for the internship, type of optimization that will be suitable to use in inferencing the computer vision model used by the organization.

GOALS OF THE INTERNSHIP

The goals of the internship were as follows:

- To benchmark inference performance of a specific Computer Vision Model being used at the organization via various inference frameworks based on throughput, latency, and power consumption at varying batch sizes
- To utilize following frameworks for inference: Tensorflow (Tf), TensorflowXLA (Tfxla), Keras (Keras), KerasXLA (Kerasxla), TensorRT FP16 Precision (Tftrtfp16), TensorRT FP32 Precision (Tftrtfp32), ONNX Run Time (Onnxrt), Torch
- Extend the evaluation results by using different GPUs

EXPERIMENTAL SETTINGS

a) **Machine 1** is a Nvidia A2, Ampere GPU containing 1280 Cuda cores running at 1440 MHz, 40 Tensor cores and 16G DDR6 of GPU memory also known as VRAM of GPU. This card was launched in November 2021, and has a Thermal Design Power (TDP) of 60 watts, which means the card can handle about 60W of heat dissipation. It has a compute capability of 8.6, bandwidth of 200.1 GB/s and memory bus with width of 128 bit

b) **Machine 2** is a Nvidia Tesla V100 PCIe, containing 5120 Cuda cores running at 1230 MHz, 640 Tensor cores and 32G HBM2 of GPU memory. This card was launched in March 2018, and has a TDP of 250 watts. It has a compute capability of 7.0, bandwidth of 897 GB/s and memory bus width of 4096 bit

c) **Machine 3** is a Nvidia Titan X Pascal containing 3580 Cuda cores running at 1417 MHz, no Tensor cores and 12G GDDR5X GPU memory. This card was launched in August 2016 and has a TDP of 250 watts. It has a compute capability of 6.1, bandwidth of 480.4 GB/s and memory bus width of 384 bit

Another key point is we do not use the tensor cores extensively in inference, as majorly all the models are benchmarked using the single precision i.e., FP32, it is only for Tftrtfp16, the tensor cores are being utilized. Utilizing tensor cores is solely dependent on neural layers mathematical operations, inference frameworks optimizations of half precision settings. However, it was not possible to test which frameworks use it exactly in what context for us, due to time constraints.

The system design used for deriving the results can be described via figure 1. The benchmarked system comprises of a partial portion of CPU, which was involved in onloading the data onto GPU and offloading the results from the GPU. The CPU was an integral part of the system, which took the input from the user, onloaded the input on GPU. Later GPU performed inference by running the input via the neural network and produced the results. These results were carried back to the user via CPU for interpretation.

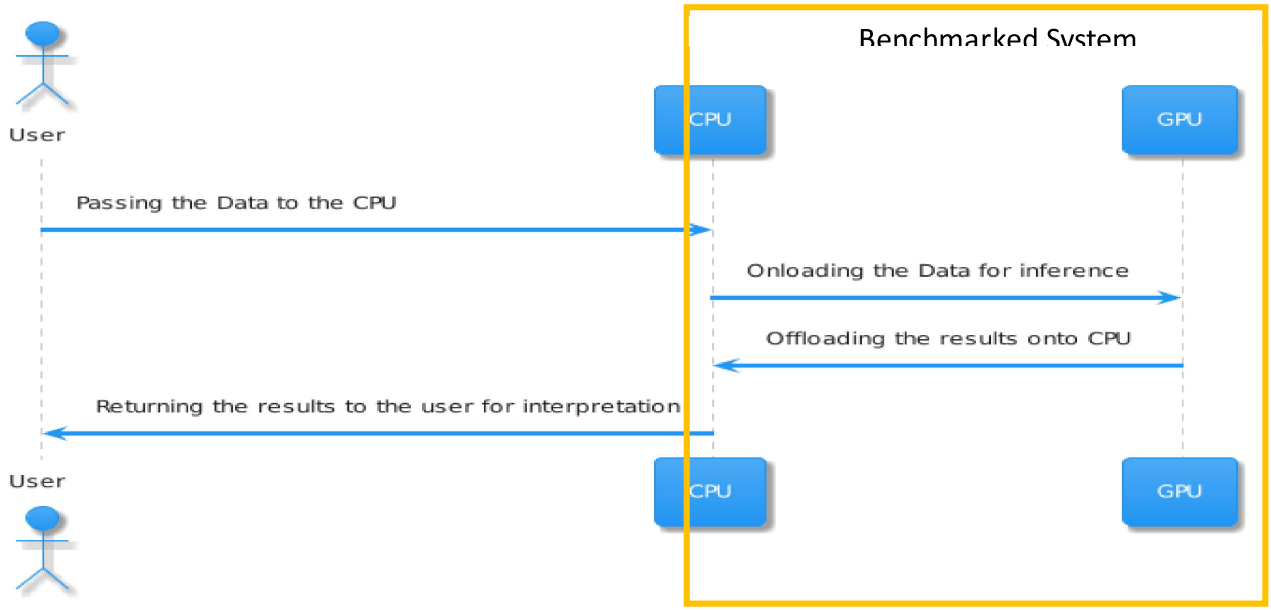


Figure 1. System Components

The system was designed to execute inference tasks based on user input, producing predictions as output. The system's parameters encompassed the hardware specifications of the GPU, while workload parameters included different batch sizes for inference tasks. Additionally, the choice of inference framework was considered a workload parameter due to its influence on benchmarking outcomes. Real-time measurements were employed to assess the system's performance, focusing on metrics such as throughput, latency, and power efficiency. Our experimental design specifically examined the effects of varying batch sizes and inference frameworks on the system's performance.

For benchmarking the system, Python APIs were created, for creating a computer vision model, and then to perform inference via different inference frameworks. How to setup the API's and run them is detailed in the GitHub repository of the project (Link is attached in the Appendix). To test the performance of a framework, we stacked an image of size 224 by 224, to create an array of size 512. Later this stack was batched in many sizes increasing in 2 folds from 8 to 512. Every framework was benchmarked on all the 7 batch sizes, except Torch which was only benchmarked on 6 batch sizes i.e., 8, 16, 32, 64, 128, 256 because, of out of memory error (OOM) for 512 batch size.

For consistent benchmarking results, 10 loops were executed for each batch size using the entire stack of 512 images. The average values of each metric across the 10 loops were calculated to represent a single value for a specific batch size withing a given framework.

Throughput was determined using the formula (total images inferred)/ (total time taken to infer) per loop, hence the number of images fixed at 512 and time varying. Latency was computed as the inverse of throughput indicating the time taken to infer a single image. Whereas throughput gave us the value of number of images inferred per second. In the results section we only discuss throughput as apart from numerical value, insights derived from both remain the same.

For assessing power efficiency, a software tool called Energy Measurement Application (EMA) developed at our Perfacct organization. This tool is based on NVML ("NVIDIA Management Library (NVML)." n.d. NVIDIA Developer. Accessed January 16, 2024) and its working was validated against a physical power consumption measuring device by developers, ensuring synchronous results. EMA automatically considers all the available hardware components of CPU and GPU. The final output of the EMA tool is by measurement region and hardware.

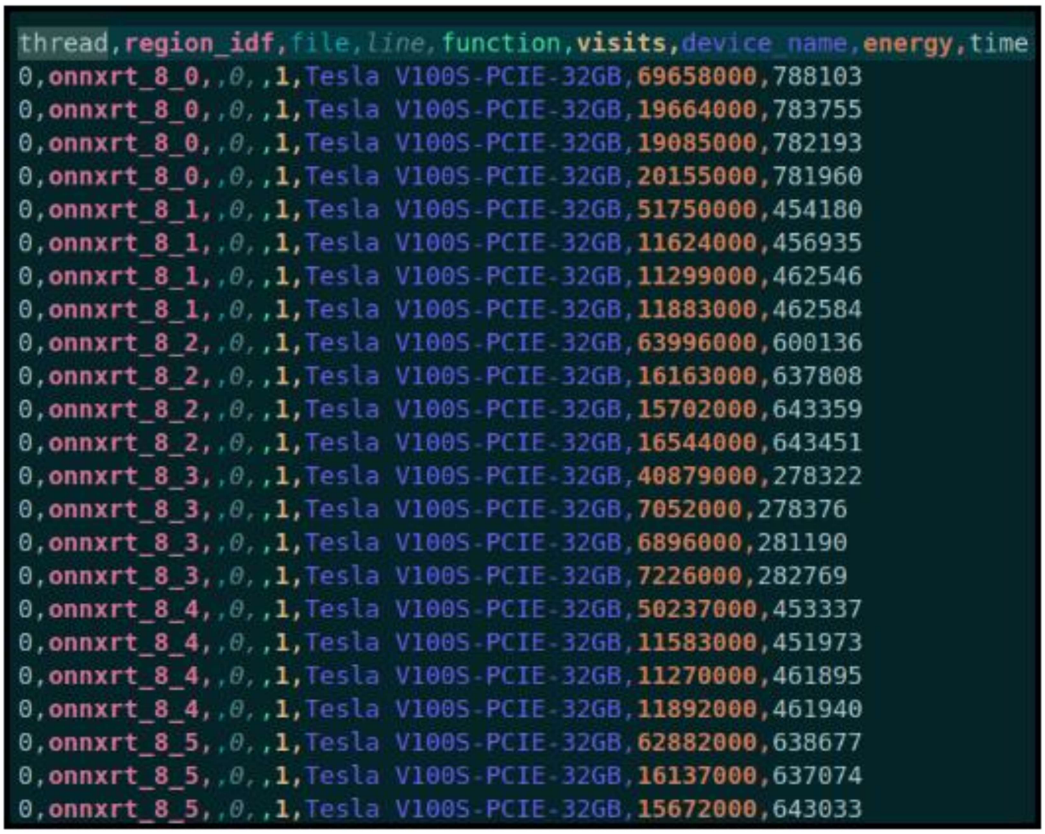
To use EMA, a user must add measurement regions in the code as shown in figure 2. For setting up EMA, instructions from the GitHub Readme could be followed up. Just one crucial instruction is

EMA_init() and EMA_finalize() must be present in the outermost code blocks if the measurement regions are added to submodules of a python file. In this project the EMA_init() and EMA_finalize() are added to the utility API script for every inference framework, whereas the measurement regions are present to the common measurement module used by all the frameworks. To differentiate between frameworks, user can add different region name for each measurement. Once the framework measurement completes its execution, EMA automatically saves a file named as <PID of the API run>.csv to present working directory (PWD). The essential columns to mark in csv are region_idf, device_name, energy, and time. Name of the region is present in the region_idf column. Energy and time are both in micro units.

```
from EMA import (
    EMA_finalize,
    EMA_init,
    EMA_region_begin,
    EMA_region_define,
    EMA_region_end,
)

# Initialization.
EMA_init()

# Create a measurement region. First argument is the region name. It will be
# used in the output.
region = EMA_region_define('region_name')

# Start measurement for region0.
EMA_region_begin(region)

# Code to be measured.
...

# Stop measurement for region0.
EMA_region_end(region)

# Cleanup.
EMA_finalize()
```

Figure 2. EMA Measurement Region Setup

Lastly the csv output is like the following figure:



```
thread,region_idf,file,line,function,visits,device name,energy,time
0,onnxrt_8_0,,0,,1,Tesla V100S-PCIE-32GB,69658000,788103
0,onnxrt_8_0,,0,,1,Tesla V100S-PCIE-32GB,19664000,783755
0,onnxrt_8_0,,0,,1,Tesla V100S-PCIE-32GB,19085000,782193
0,onnxrt_8_0,,0,,1,Tesla V100S-PCIE-32GB,20155000,781960
0,onnxrt_8_1,,0,,1,Tesla V100S-PCIE-32GB,51750000,454180
0,onnxrt_8_1,,0,,1,Tesla V100S-PCIE-32GB,11624000,456935
0,onnxrt_8_1,,0,,1,Tesla V100S-PCIE-32GB,11299000,462546
0,onnxrt_8_1,,0,,1,Tesla V100S-PCIE-32GB,11883000,462584
0,onnxrt_8_2,,0,,1,Tesla V100S-PCIE-32GB,63996000,600136
0,onnxrt_8_2,,0,,1,Tesla V100S-PCIE-32GB,16163000,637808
0,onnxrt_8_2,,0,,1,Tesla V100S-PCIE-32GB,15702000,643359
0,onnxrt_8_2,,0,,1,Tesla V100S-PCIE-32GB,16544000,643451
0,onnxrt_8_3,,0,,1,Tesla V100S-PCIE-32GB,40879000,278322
0,onnxrt_8_3,,0,,1,Tesla V100S-PCIE-32GB,7052000,278376
0,onnxrt_8_3,,0,,1,Tesla V100S-PCIE-32GB,6896000,281190
0,onnxrt_8_3,,0,,1,Tesla V100S-PCIE-32GB,7226000,282769
0,onnxrt_8_4,,0,,1,Tesla V100S-PCIE-32GB,50237000,453337
0,onnxrt_8_4,,0,,1,Tesla V100S-PCIE-32GB,11583000,451973
0,onnxrt_8_4,,0,,1,Tesla V100S-PCIE-32GB,11270000,461895
0,onnxrt_8_4,,0,,1,Tesla V100S-PCIE-32GB,11892000,461940
0,onnxrt_8_5,,0,,1,Tesla V100S-PCIE-32GB,62882000,638677
0,onnxrt_8_5,,0,,1,Tesla V100S-PCIE-32GB,16137000,637074
0,onnxrt_8_5,,0,,1,Tesla V100S-PCIE-32GB,15672000,643033
```

Figure 3. EMA Measurements

The outcome of the benchmarks is detailed in the subsequent section.

RESULTS & READING CONVENTIONS

We present the results in tables in the appendix. Appendix Table A.1 denotes the results for throughput, whereas table A.2 denotes the results for power consumption. In this section we present the convention to read the result tables while we discuss them in the next section. We also add the table for a single batch size here for better understanding.

For reading the throughput table, some conventions need to be kept in mind:

a) Every cell on the table has a gradient background going from orange to green. Within a specific batch size, a greener cell indicates a higher throughput, with the highest throughputs being the darkest green
b) The gradient colors for a particular batch size do not consider the throughputs for other batch sizes. For example, in batch size 8 the highest throughput of 1800 will be the darkest, whereas in batch size of 32 it can be lighter, as there is no relation between the 2 batch sizes
c) Bold numbers signify the optimal framework for a GPU column, across various batch sizes
d) GPUs that outperformed the V100 in terms of throughput for a specific inference framework are highlighted with an asterisk (*)

	Py Throughput Comparison			
GPU Name / Batch Size	Ampere_A2	Volta_100	Titan_X	Frameworks
16	109.405756	149.523644	146.824404	Tf
16	785.545358	851.564389	*991.566910	Tfxla
16	114.567435	149.399137	149.186473	Keras
16	786.159522	842.484617	*989.880349	Kerasxla
16	**857.992149**	753.651050	*1221.707180	Tftrtfp16
16	669.306551	669.417060	*1100.895615	Tftrtfp32
16	674.925308	**1581.245670**	**1311.685751**	Onnxrt
16	598.476632	1304.533150	1017.472303	Torch

Table 1. Sample Throughput Table

For understanding the power consumption table, the following guidelines can be followed:

a) Like throughput, the shading for power consumption progresses from green to orange for every batch size. A greener cell within a batch size indicates lower power consumption, with the lowest consumption appearing as the darkest green
b) As with the throughput table, the shading for a specific batch size does not consider power consumptions for other batch sizes
c) Bold numbers indicate the best framework for a GPU across different batch sizes

	Py Power (Watts) Comparison			
GPU Name / Batch Size	Ampere_A2	Volta_100	Titan_X	Frameworks
16	**26.842600**	**44.739453**	**72.713509**	Tf
16	43.272013	85.508961	160.557990	Tfxla
16	28.468067	45.772509	77.255074	Keras
16	44.946666	73.979140	174.491715	Kerasxla
16	31.189324	56.963799	160.030168	Tftrtfp16
16	35.638492	70.204338	144.228988	Tftrtfp32
16	49.737854	126.537680	226.617048	Onnxrt
16	47.257480	106.988506	187.909724	Torch

Table 2. Sample Power Consumption Table

DISCUSSION

From the conducted analysis following conclusions can be drawn:

Throughput (Appendix Table A.1)

- On machine 1 of Ampere A2, majorly the TensorRT frameworks (Tftrtfp16 and fp32, provide the best throughputs from the batch size of 16 to 256, while Onnxrt seems to outperform on batch size of 8 and 512.
- On machine 2 of Nvidia Tesla V100, majorly Onnxrt framework outperforms every other inference framework, by significant margins. Torch framework seems to keep up well alongside after Onnxrt on this GPU.
- On machine 3 of Nvidia Titan X Pascal, oldest of the 3, Onnxrt framework outperformed every other inference framework on every batch size except batch size of 32. Tfxla outperformed other frameworks for the batch size of 16.
- While looking at the results across GPUs for comparison when one specific batch size and framework is selected, we found out that, Nvidia Tesla V100 outperformed the other 2 GPUs. It is only for the smaller batch sizes from 8 to 32, where the Titan X Pascal outperformed V100, on some XLA and TensorRT based frameworks.
- Few more conclusions we were able to draw during the analysis are:
    - More Cuda cores can draw more throughput out of a GPU
    - Only having a higher base clock operations speed is not enough for higher throughputs
    - Having high compute capability is only useful if a model uses those operations that come bundled with high compute capability.
    - Lastly for higher throughput, we saw having bigger bus width and bandwidth can help.

Latency

Since latency is simply the inverse of throughput, we will not be discussing those, as the insights remain in line with throughput, only change of numerical values will occur.

Power Consumption (Appendix Table A.2)

- From the analysis, we were able to conclude that using newer GPU is much more power efficient as compared to using older GPUs. The older the GPU is the less power efficient it is.
- Machine 1 of Ampere A2, in every case showed better power consumption in watts, as compared to the other GPU's.
- On each of the machines till batch size of 32, Tf framework was the most power efficient. However, as the batch size starts to increase 32 onwards the best power efficient framework is either Tftrtfp16 or Tftrtfp32. The values of Tftrt are comparable, with a difference of 20 to 30 watts in power consumption.
- The last conclusion from the analysis was that Onnxrt, even though one of the best performing frameworks in terms of throughput, is not very power consumption efficient.

Lastly from the analysis, we thought it would be meaningful to see if a pattern exists between power consumption and throughput. Following plots highlight this relationship and we can observe a general trend in every GPU that, for higher throughput there is a higher power consumption. Apart from this the R Squared values denote that with throughput how well we can predict power consumption. The higher the value of R Square the perfect the prediction of power consumption with throughput. An ideal framework will be the line with a negative slope, which yields high throughput with less power consumption. By analyzing the plots, we concluded that the frameworks, which show a less positive slope, could be useful for real-world scenario as the power consumption does not scale very rapidly with increase in throughput. From the analysis the good frameworks depending on the GPU will be:

a) On Ampere A2 using Tftrtfp16, Tftrtfp32 and KerasXla might be beneficial
b) On Tesla V100 using Tftrtfp32 might be beneficial
c) On Titan X using Tftrtfp16, Tftrtfp32 and Onnxrt might be efficient

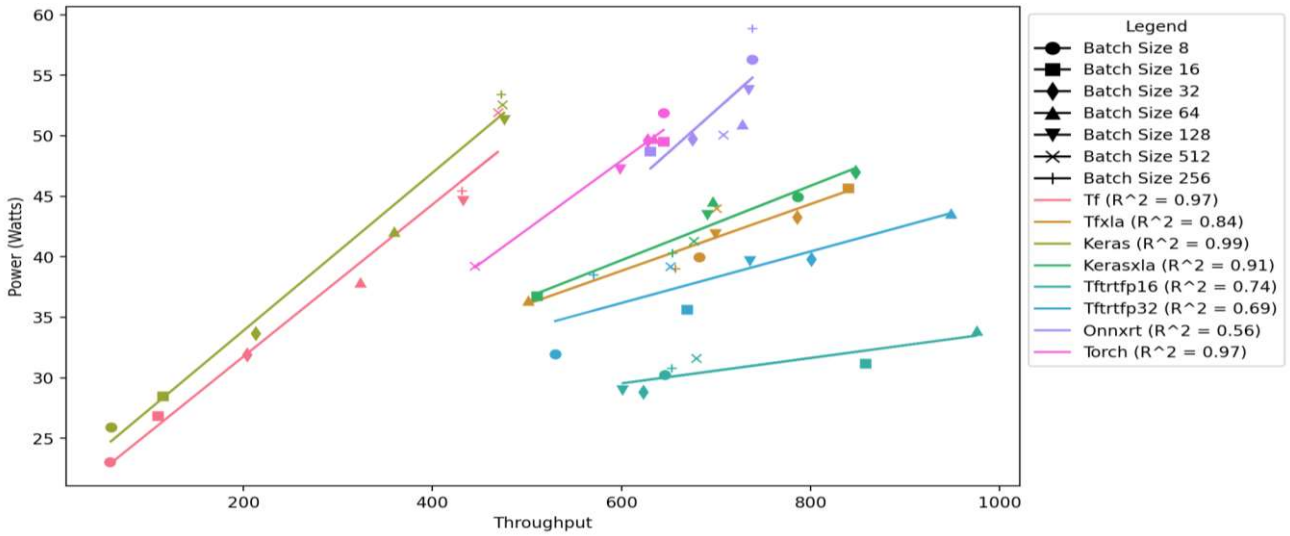


Figure 4. Power vs Throughput on Ampere GPU

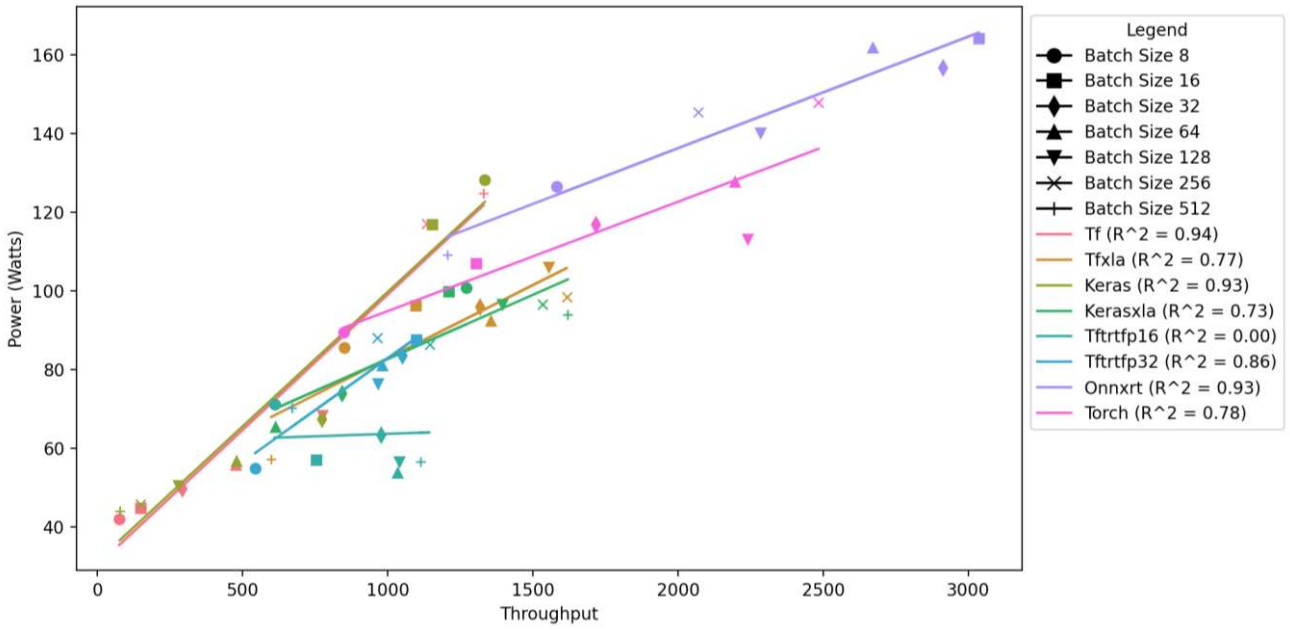


Figure 5. Power vs Throughput on Nvidia Tesla V100 GPU

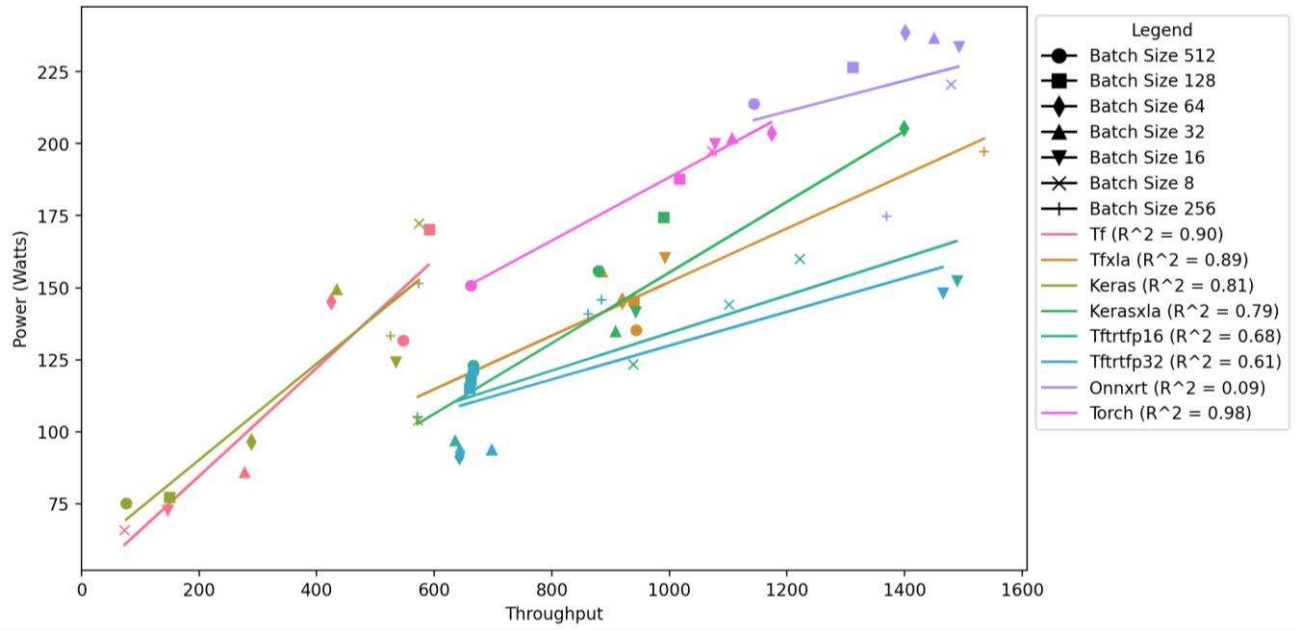


Figure 6. Power vs Throughput on Titan X Pascal GPU

CONCLUSION

From the analysis, for the organization, using tensorRT might be most beneficial given the use to use 32 or 64 as the batch size and on these batch sizes, tensorRT delivers better results in terms of throughput as compared to other frameworks. Along with this, the power consumption of tensorRT is also sustainable. Lastly to add up on the future works, we believe benchmarking the models' accuracy with the frameworks, testing more intricate optimizations within the frameworks, utilization of single precision and mixed precision capabilities of frameworks will be possible.

REFERENCES

1) Liu, Ling, Yanzhao Wu, Wenqi Wei, Wenqi Cao, Semih Şahin, and Qi Zhang. 2018. "Benchmarking Deep Learning Frameworks: Design Considerations, Metrics and Beyond." In, 1258–69. https://doi.org/10.1109/ICDCS.2018.00125.

2) Pochelu, Pierrick. 2022. "Deep Learning Inference Frameworks Benchmark." arXiv. http://arxiv.org/abs/2210.04323.

3) Liang, Tailin, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. "Pruning and Quantization for Deep Neural Network Acceleration: A Survey." arXiv.Org. January 24, 2021. https://arxiv.org/abs/2101.09671v3.

4) Zhou, Yuxiao, and Kecheng Yang. 2022. "Exploring TensorRT to Improve Real-Time Inference for Deep Learning." In 2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), 2011–18. https://doi.org/10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00299.

5) "56ea2bb991a7446776ac2f2f27fdc397.Jpg (618×373)." n.d. Accessed December 11, 2024. https://i.pinimg.com/originals/56/ea/2b/56ea2bb991a7446776ac2f2f27fdc397.jpg.

6) “NVIDIA Management Library (NVML).” n.d. NVIDIA Developer. Accessed January 15, 2024. https://developer.nvidia.com/nvidia-management-library-nvml.

APPENDIX

GitHub_Project_Link: Perfacct_Internship (https://github.com/dhepeyuvi/Perfacct_Internship)

	Throughput Comparison			
GPU Name / Batch Size	Ampere_A2	Tesla V100	Titan_X	Frameworks
8	58.297133	75.604844	72.878696	Tf
8	501.147405	599.020824	571.730992	Tfxla
8	59.718775	77.355360	75.926725	Keras
8	510.339839	612.398705	571.174690	Kerasxla
8	622.697064	610.407696	883.974033	Tftrtfp16
8	529.840505	543.701020	861.011500	Tftrtfp32
8	**630.387213**	**1206.454856**	**1143.399651**	Onnxrt
8	444.858403	849.322592	661.773001	Torch
16	109.405756	149.523644	146.824404	Tf
16	785.545358	851.564389	*991.566910	Tfxla
16	114.567435	149.399137	149.186473	Keras
16	786.159522	842.484617	989.880349	Kerasxla
16	**857.992149**	753.651050	*1221.707180	Tftrtfp16
16	669.306551	669.417060	*1100.895615	Tftrtfp32
16	674.925308	**1581.245670**	**1311.685751**	Onnxrt
16	598.476632	1304.533150	1017.472303	Torch
32	203.784868	291.201569	277.623951	Tf
32	839.797382	1097.242220	***1534.981237**	Tfxla
32	213.048204	279.447808	*288.934885	Keras
32	847.485870	1208.999935	*1398.718157	Kerasxla
32	**975.476683**	977.806913	*1488.889271	Tftrtfp16
32	800.622208	964.374992	*1465.356610	Tftrtfp32
32	707.642532	**2069.762996**	1401.301924	Onnxrt
32	633.768301	1716.332483	1173.472941	Torch
64	323.738303	476.358691	424.542882	Tf

64	657.069909	1316.556092	885.678998	Tfxla
64	359.321834	478.643029	433.481476	Keras
64	653.231476	1270.057756	878.987318	Kerasxla
64	601.057097	1032.467878	635.544339	Tftrtfp16
64	**948.540819**	966.914429	696.914507	Tftrtfp32
64	727.829062	**2284.628594**	**1450.062727**	Onnxrt
64	627.214559	2195.434873	1106.403052	Torch
128	432.190862	774.316719	591.062957	Tf
128	682.096063	1355.950843	919.432857	Tfxla
128	475.889428	772.513689	574.025660	Keras
128	676.103249	1393.886717	907.751861	Kerasxla
128	645.939720	1039.567566	642.907215	Tftrtfp16
128	**735.877531**	980.794765	643.403225	Tftrtfp32
128	734.561164	**2670.985313**	**1492.067404**	Onnxrt
128	644.121230	2240.666185	1077.002209	Torch
256	430.931939	1134.484002	572.893619	Tf
256	698.867848	1555.520219	939.552513	Tfxla
256	473.986340	1152.850980	534.509401	Keras
256	690.966607	1533.208035	941.828968	Kerasxla
256	652.720480	1112.812002	661.633062	Tftrtfp16
256	651.665106	1050.592235	659.617338	Tftrtfp32
256	**738.270422**	**2911.127940**	**1478.806160**	Onnxrt
256	644.435542	2483.357382	1072.290332	Torch
512	469.236217	1329.551714	547.203641	Tf
512	700.271098	1617.248524	943.248448	Tfxla
512	472.887208	1334.651630	524.885081	Keras
512	696.709473	1620.324001	937.706861	Kerasxla
512	678.743574	1143.989980	666.383136	Tftrtfp16
512	570.466144	1098.806226	665.952338	Tftrtfp32
512	**738.369238**	**3036.471735**	**1368.341322**	Onnxrt

Table A.1: Throughput Comparison across GPUs & Frameworks

	Power (Watts) Comparison			
GPU Name	Ampere_A2	Volta_100	Titan_X	Frameworks

Batch Size				
8	**23.034441**	**42.028326**	**65.791985**	Tf
8	36.377593	57.266119	103.942459	Tfxla
8	25.901741	44.040553	75.208886	Keras
8	36.760216	65.447438	105.314046	Kerasxla
8	28.811368	71.073366	145.946888	Tftrtfp16
8	31.944471	54.813864	140.982488	Tftrtfp32
8	48.723148	109.124212	213.813379	Onnxrt
8	39.262233	89.478817	150.696216	Torch
16	**26.842600**	**44.739453**	**72.713509**	Tf
16	43.272013	85.508961	160.557990	Tfxla
16	28.468067	45.772509	77.255074	Keras
16	44.946666	73.979140	174.491715	Kerasxla
16	31.189324	56.963799	160.030168	Tftrtfp16
16	35.638492	70.204338	144.228988	Tftrtfp32
16	49.737854	126.537680	226.617048	Onnxrt
16	47.257480	106.988506	187.909724	Torch
32	**31.915847**	**49.743864**	**86.051653**	Tf
32	45.679376	96.202626	197.270532	Tfxla
32	33.665447	50.533698	96.533293	Keras
32	47.007559	99.862280	205.441184	Kerasxla
32	33.889369	63.449932	152.449504	Tftrtfp16
32	39.794271	88.127034	148.186995	Tftrtfp32
32	50.096332	145.416263	238.536977	Onnxrt
32	49.761281	116.940983	203.740613	Torch
64	37.877169	55.855312	145.293781	Tf
64	39.048159	96.006308	155.570689	Tfxla
64	42.083285	56.774347	149.542282	Keras
64	40.323288	100.718072	155.797082	Kerasxla
64	**29.016006**	**53.788084**	96.986876	Tftrtfp16
64	43.572691	76.380378	**93.818586**	Tftrtfp32
64	50.965443	140.109348	236.822147	Onnxrt
64	49.597481	127.807230	201.962507	Torch
128	44.662014	68.347610	170.184689	Tf
128	39.956464	92.384316	145.488574	Tfxla
128	51.345073	67.421122	172.404259	Keras
128	41.308315	96.549081	134.855891	Kerasxla

128	**30.231717**	**56.569295**	**91.383410**	Tftrtfp16
128	39.662400	81.083141	92.855731	Tftrtfp32
128	53.836334	161.858718	233.783093	Onnxrt
128	49.525353	113.132439	200.137332	Torch

256	45.473414	117.041113	151.657132	Tf
256	41.918893	106.124414	145.554073	Tfxla
256	52.595597	116.912042	124.364485	Keras
256	43.469974	96.614300	141.733747	Kerasxla
256	**30.786251**	**56.503715**	117.892633	Tftrtfp16
256	39.220769	83.547054	**115.069574**	Tftrtfp32
256	56.311575	156.775507	220.546182	Onnxrt
256	51.910435	147.977753	197.626068	Torch

512	51.945824	124.888077	131.757324	Tf
512	44.000078	98.443944	135.369418	Tfxla
512	53.449223	128.229717	133.372762	Keras
512	44.561657	93.956220	123.388569	Kerasxla
512	**31.630068**	**86.406511**	122.948484	Tftrtfp16
512	38.505054	87.668504	**120.837024**	Tftrtfp32
512	58.857947	164.172597	174.797420	Onnxrt

Table A.2: Power Consumption Comparison across GPUs & Frameworks