Applied Machine Learning in the Cloud
Jessica Liang

Experiment Design
The objective of this experiment is to analyze the performance of training neural networks in a virtual machine on google cloud versus in a container on google cloud. The performance metrics are not based on accuracy; rather, they are based on GPU utilization, memory bandwidth, and FLOPS utilization.

Complexity Estimation
My hypothesis was that training the networks directly on the VM will result in optimized performance in terms of GPU usage, memory bandwidth, and FLOP usage. I theorize that a containerized workload would introduce another layer of abstraction that the system must take on.

I used the tiny-imagenet-200 dataset to train the models and stopped after 15 batches. I trained it on these following networks: resnet18, resnet50, and alexnet. For additional investigation, my hypothesis was that the computational cost of resent18 would be significantly less than ResNet50, since there are significantly less layers (almost one-third less), but I was interested to see how alexnet would perform compared to resnet18. Online literature shows conflicting results, and I was intrigued by how effective skipping connections, which resnet18 implements, could impact training.

Measurement
First, using a modified script from last homework, I found a region with a GPU available and created it. I cloned the pytorch repository for training models, as well as the dataset in the VM. This allowed me to test the training of different models with the VM.

I wrote a script to track the methods for efficiency, which called the cloned pytorch training code. The script loops through each model and stores the results in a separate directory.

For the containerized method, I created a Docker image on my local machine and pushed it to another VM, which I found a GPU in with the same python script. This allows for performance metrics using the container method.

One downside to this is it took 199.7s to build, and then additional time to push the image because it was almost 5GB. I also had to rebuild many times locally on a Mac for a Linux machine before building it directly on the VM. More space was necessary for the first method.

```
(base) jessicaliang@MacBookAir project1 % docker build -t imagenet_training:latest -f Dockerfile .

[+] Building 199.7s (17/17) FINISHED                                    docker:desktop-linux
```

```
(base) jyl2196@containter-gpu-test:~$ docker images
REPOSITORY                          TAG       IMAGE ID       CREATED          SIZE
jessicayliang/imagenet_training     latest    29d0ca4c95d1   5 minutes ago    4.91GB
hello-world                         latest    74cc54e27dc4   2 months ago     10.1kB
```

Roofline Modeling

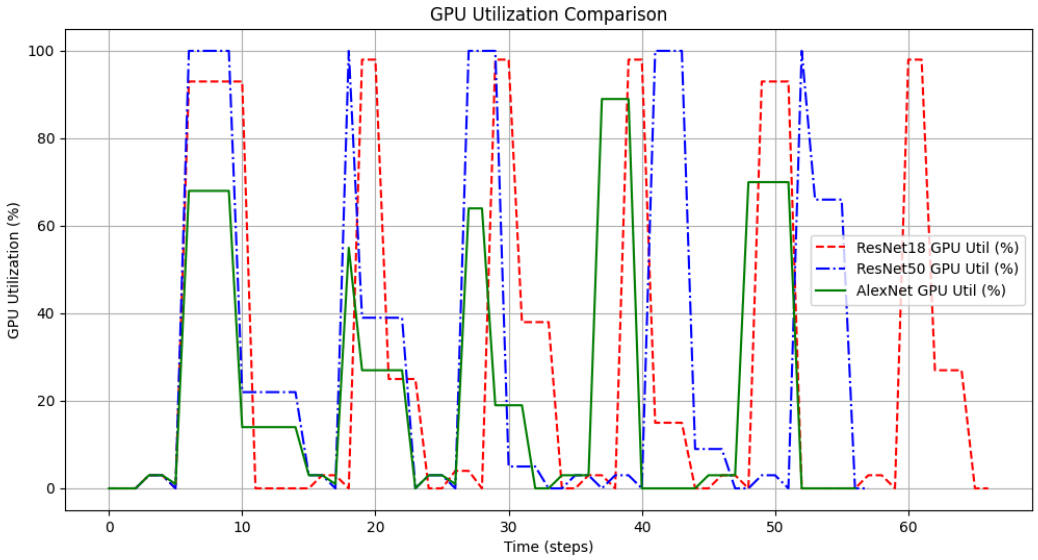

Figure 1: GPU utilization of ResNet18, ResNet50, and AlexNet directly on the VM.

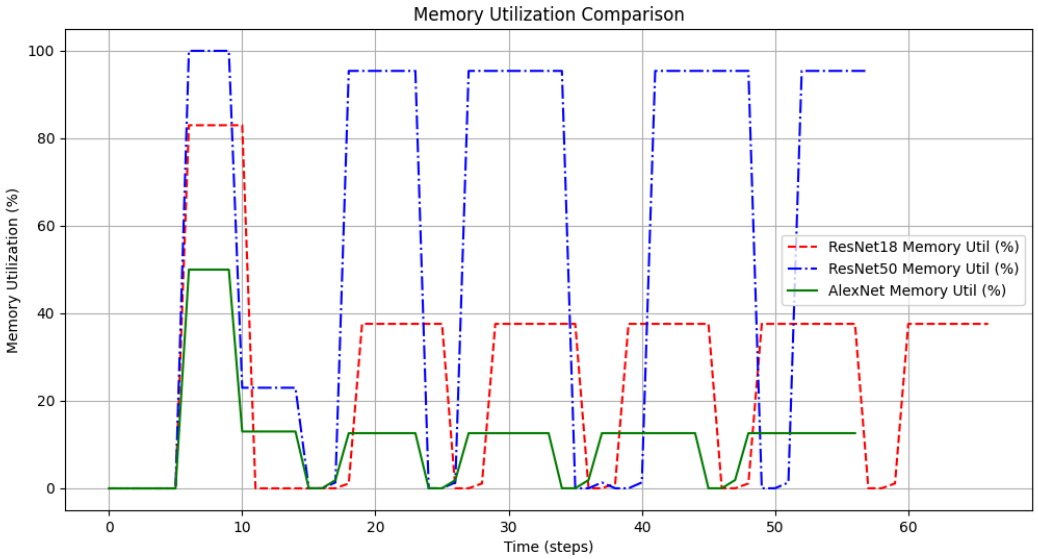

Figure 2: Memory utilization of ResNet18, ResNet50, and AlexNet directly on the VM.

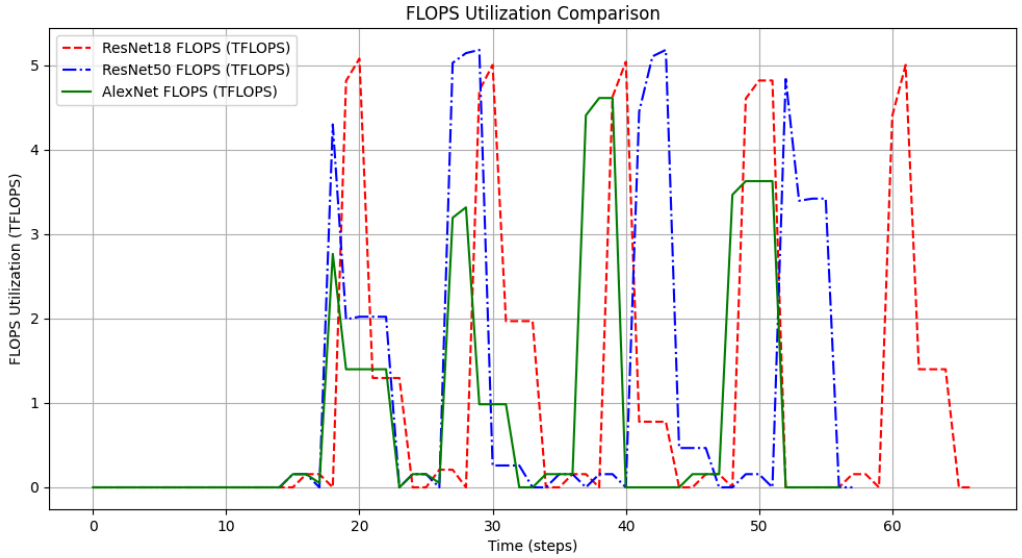

Figure 3: FLOPS utilization of ResNet18, ResNet50, and AlexNet directly on the VM.

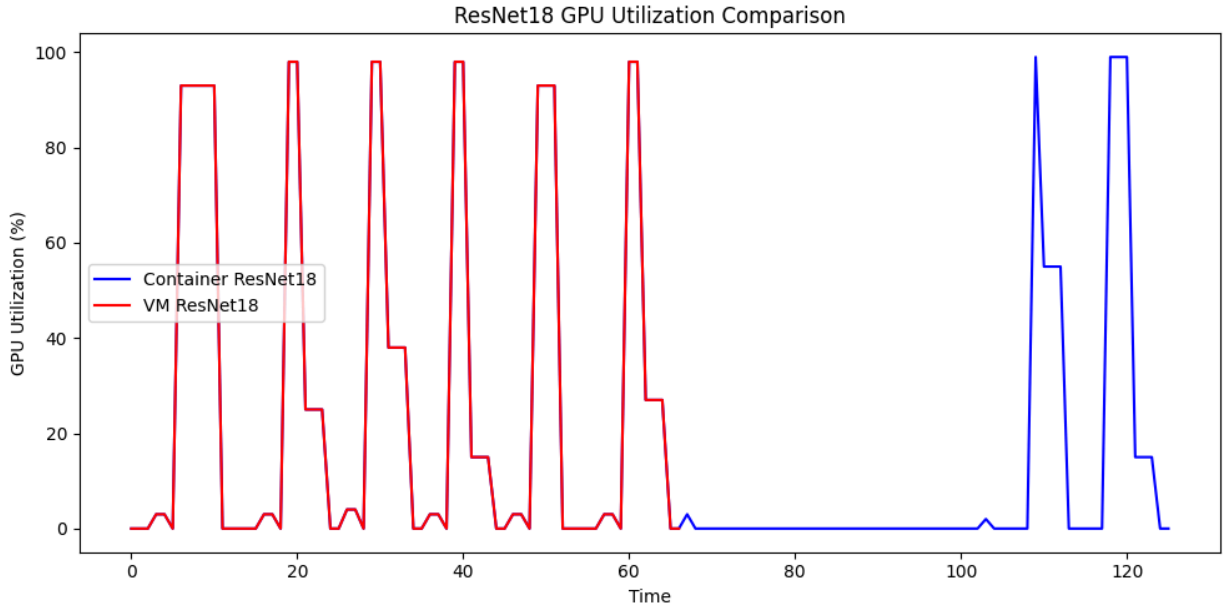

Figure 4: ResNet18 GPU Utilization on container, side-by-side with utilization directly on a VM.

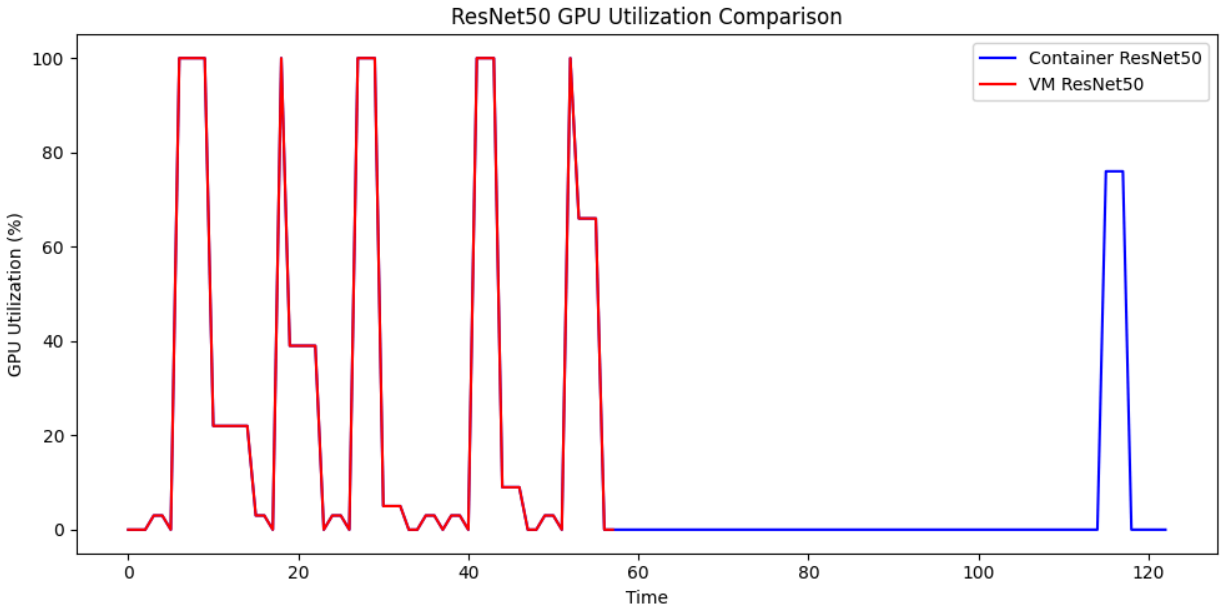

Figure 5: ResNet50 GPU Utilization on container, side-by-side with utilization directly on a VM.

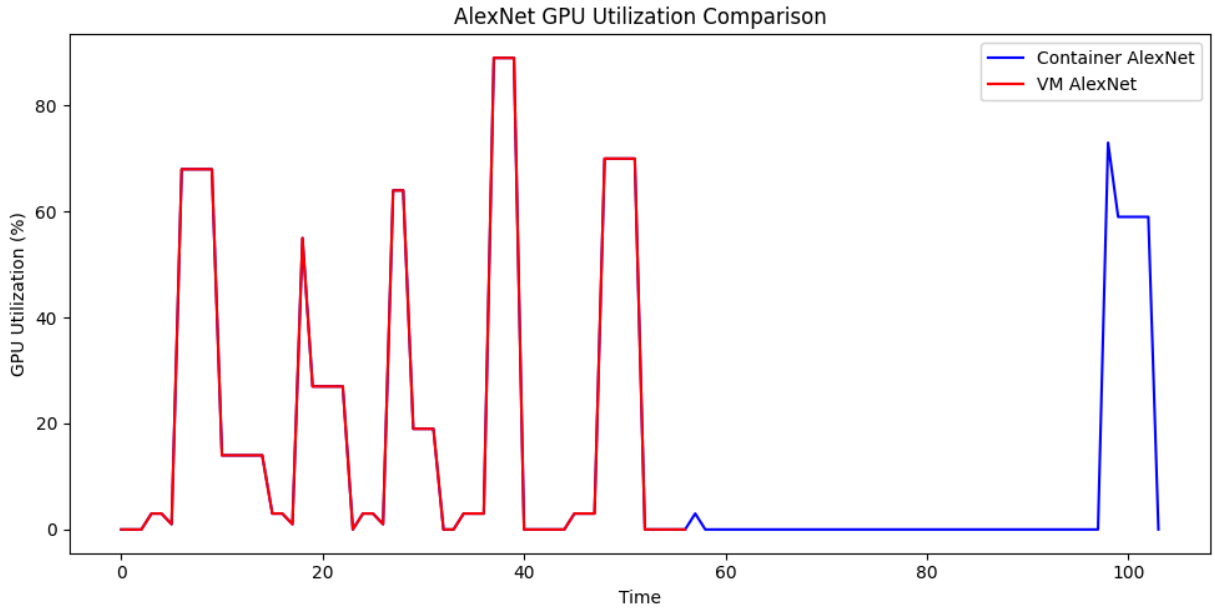

Figure 6: AlexNet GPU Utilization on container, side-by-side with utilization directly on a VM.

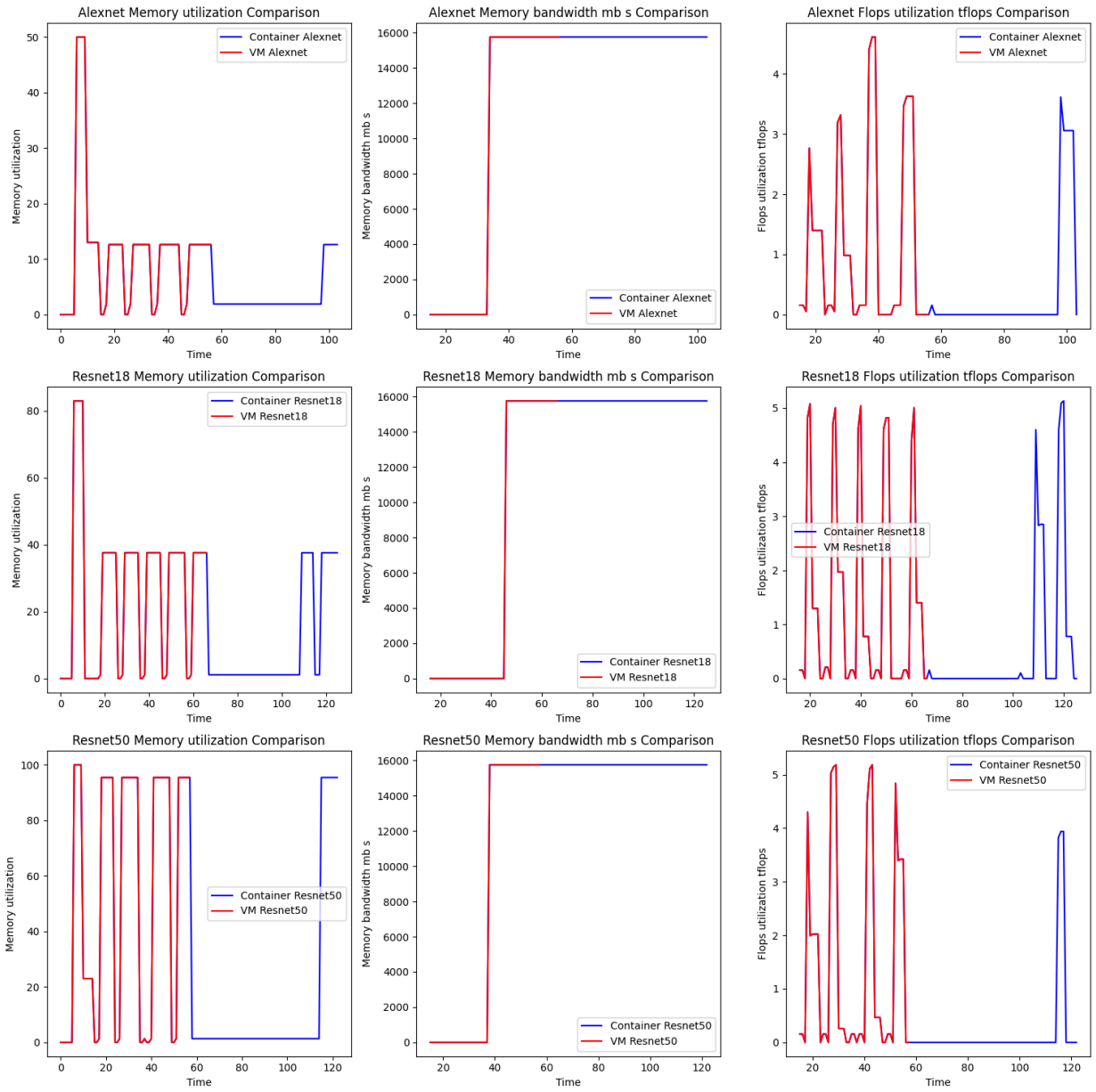

Figure 7: Comparisons of ResNet18, ResNet50, and AlexNet running on a container in a VM versus directly on the VM using memory utilization, memory bandwidth, and FLOPs.

```
/Users/jessicaliang/.pyenv/versions/3.9.13/bin/python3 /Users/jessicaliang/Desktop/project1 graphs/metrics.py
Container AlexNet Max Metrics: {'max_gpu_utilization': 89.0, 'max_memory_bandwidth_mb_s': 15760.0, 'max_flops_utilization_tflops': 4.61376}
VM AlexNet Max Metrics: {'max_gpu_utilization': 89.0, 'max_memory_bandwidth_mb_s': 15760.0, 'max_flops_utilization_tflops': 4.61376}
Container ResNet18 Max Metrics: {'max_gpu_utilization': 99.0, 'max_memory_bandwidth_mb_s': 15760.0, 'max_flops_utilization_tflops': 5.13216}
VM ResNet18 Max Metrics: {'max_gpu_utilization': 98.0, 'max_memory_bandwidth_mb_s': 15760.0, 'max_flops_utilization_tflops': 5.08032}
Container ResNet50 Max Metrics: {'max_gpu_utilization': 100.0, 'max_memory_bandwidth_mb_s': 15760.0, 'max_flops_utilization_tflops': 5.184}
VM ResNet50 Max Metrics: {'max_gpu_utilization': 100.0, 'max_memory_bandwidth_mb_s': 15760.0, 'max_flops_utilization_tflops': 5.184}

Process finished with exit code 0
```

Figure 8: Maximum value of metrics from python script (transcribed to table.

Below is a table for the highest values of each metric to compare the VM and the container in a VM.

	VM on GCP	Container in VM on GCP
resnet18	max_gpu_utilization': 98.0, 'max_memory_bandwidth_mb_s': 15760.0, 'max_flops_utilization_tflops': 5.08032	max_gpu_utilization': 99.0, 'max_memory_bandwidth_mb_s': 15760.0, 'max_flops_utilization_tflops': 5.13216
resnet50	max_gpu_utilization': 100.0, 'max_memory_bandwidth_mb_s': 15760.0, 'max_flops_utilization_tflops': 5.184	max_gpu_utilization': 100.0, 'max_memory_bandwidth_mb_s': 15760.0, 'max_flops_utilization_tflops': 5.184
alexnet	max_gpu_utilization': 89.0, 'max_memory_bandwidth_mb_s': 15760.0, 'max_flops_utilization_tflops': 4.61376	max_gpu_utilization': 89.0, 'max_memory_bandwidth_mb_s': 15760.0, 'max_flops_utilization_tflops': 4.61376

Table 1: Maximum metrics for ResNet18, ResNet50, and AlexNet when running directly on a VM versus in a container in a VM.

Analysis

Clearly, the most complex model (ResNet50) utilized the most resources, and the least complex model (AlexNet) utilized the least. However, there are instances in GPU and FLOPS utilization where the more complex models with layers are almost at par with each other, in terms of usage (Figure 2).

While containers were more efficient, the additional work of containerizing the model training may possibly outweigh the benefits for training a model just once. However, for repeated training, its efficiency and lightweight deployment on any machine is better than running directly on a VM—it is portable and flexible. Figures 4, 5, and 6 demonstrate how GPU utilization when training directly in a VM is not as inconsistent as when running in a container. Figure 7 demonstrates that training in the container is noticeably lower in FLOPs.

It is important, however, to consider the following for containers: space in the container, time for building images, different machine configurations (ex: mac vs. linux), and the additional work of using another platform.

Conclusion

The hypothesis was not correct, as it is evident that the impact of containerization is minimal. However, there are other trade-offs of additional labor for configuring Docker.

For repeated training, the flexibility of containers is the clear winner—they are portable and suitable for scalability.