A Study of Optimization Techniques for Scalable Image Classification Inference on the Cloud

Project's GitHub Repository

Applied Machine Learning for Cloud Computing - Final Project Report

Timothy Chang	Tyler Chang	Athitheya Gobinathan	Pablo Ordorica Wiener
tyc2118	tc3407	asg2278	po2311

Project Overview

For our project, we set out to evaluate how different post-training optimization strategies-like pruning, sparsification, and quantization-impact the latency and user experience of image classification inference on the cloud. We built on our earlier experience with Kubeflow on Google Cloud (GCP), deploying four endpoints on Google Cloud's Vertex AI: one baseline and three with distinct optimizations. To make benchmarking accessible, we developed a web app that lets users interact with each endpoint and observe the impact of these optimizations. Through this, we gained insights into the trade-offs between model performance and user experience, and we identified real-world challenges in deploying optimized AI models at scale.

Course Relevance

Our work directly relates to the course's core topics:

- **Cloud:** We implemented our ML platform on Google Cloud Platform, leveraging managed services like Vertex AI and Triton Inference Server.
- **Deep Neural Networks:** We applied post-training optimizations (pruning, sparsification, quantization) to a pre-trained ResNet50 deep learning model.
- **Performance Analysis:** We measured and compared end-to-end latency and inference speed for each deployment.

Task Distribution:

As shown in Figure 1, Tyler and Pablo focused on the ML platform, ML deployment and web app, while Athith and Tim worked on model optimization.

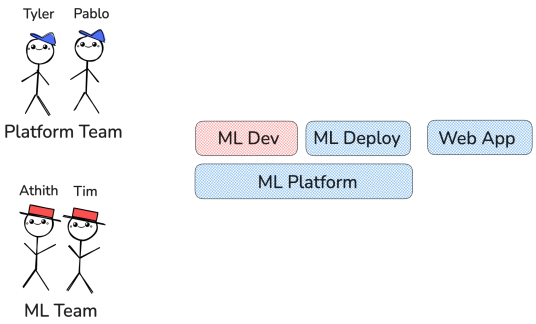

Figure 1. Task Distribution

Related Work

Our approach was informed by:

- Prior coursework on ML workflow orchestration with Kubeflow.
- Industry practices using NVIDIA Triton Inference Server for scalable model serving.
- Research and presentations on neural network optimization frameworks, including TensorRT and quantization techniques.
- Recent surveys and guides on AI inference infrastructure and production deployment.

Design & Implementation

Our end-to-end workflow for a single model followed this sequence, illustrated in Figure 2:

1. Convert pretrained ResNet50 to ONNX format
2. Apply TensorRT optimizations (pruning, sparsification, or quantization)
3. Package optimized model with Triton Inference Server in Docker
4. Deploy container to Vertex AI endpoint
5. Enable real-time inference through Streamlit web app

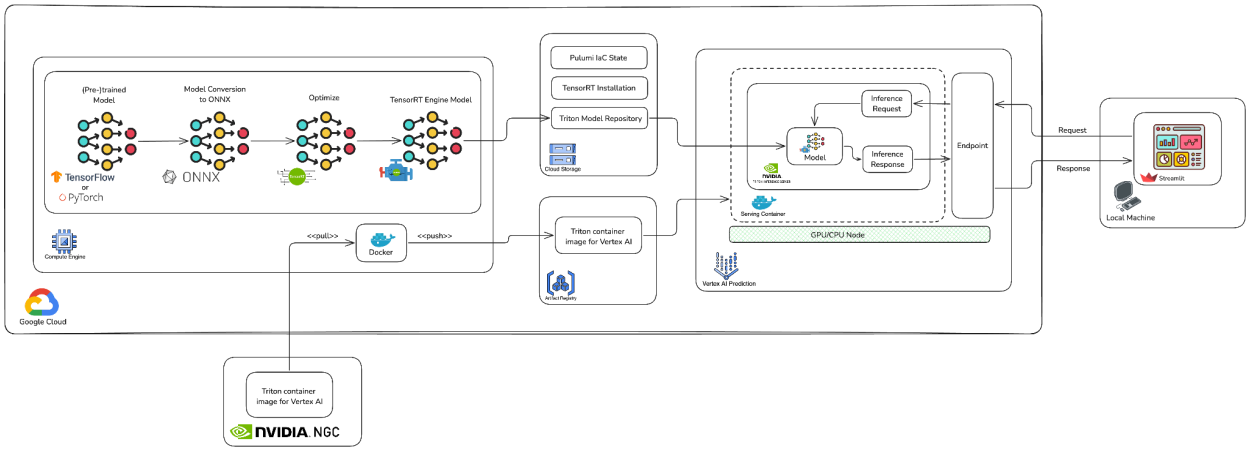

Figure 2. End-to-end Implementation for a Single Model

ML Platform Architecture

To enable the end-to-end shown in Figure 2 for four models, we built our ML Platform on Google Cloud. Our platform architecture consisted of three main infrastructure stacks, each mapped to a key phase of the end-to-end workflow.

- **Pulumi IaC Backend Stack:** Since we used Pulumi IaC for infrastructure-as-code to provision and manage the Google Cloud resources of the ML Dev Stack described next. Pulumi's state was stored in a dedicated Cloud Storage bucket, ensuring reproducibility and safe collaboration across the team. This stack only manages that bucket.
- **ML Dev Stack:** This stack provided an NVIDIA L4 GPU-enabled VM with Ubuntu 20.04, configured via Ansible, for model development and optimization. Here, we performed the model optimizations, and managed model artifacts. The stack additionally had a Cloud Storage bucket for the model repository, a bucket for the TensorRT local installation file and an artifact registry repository to store the Triton Inference Server container image that would run the models during inference.
- **ML Deployment Stack:** The ML Deployment stack automated the registration of models, creation of endpoints, and scaling of inference workloads, abstracting away the underlying infrastructure complexity.

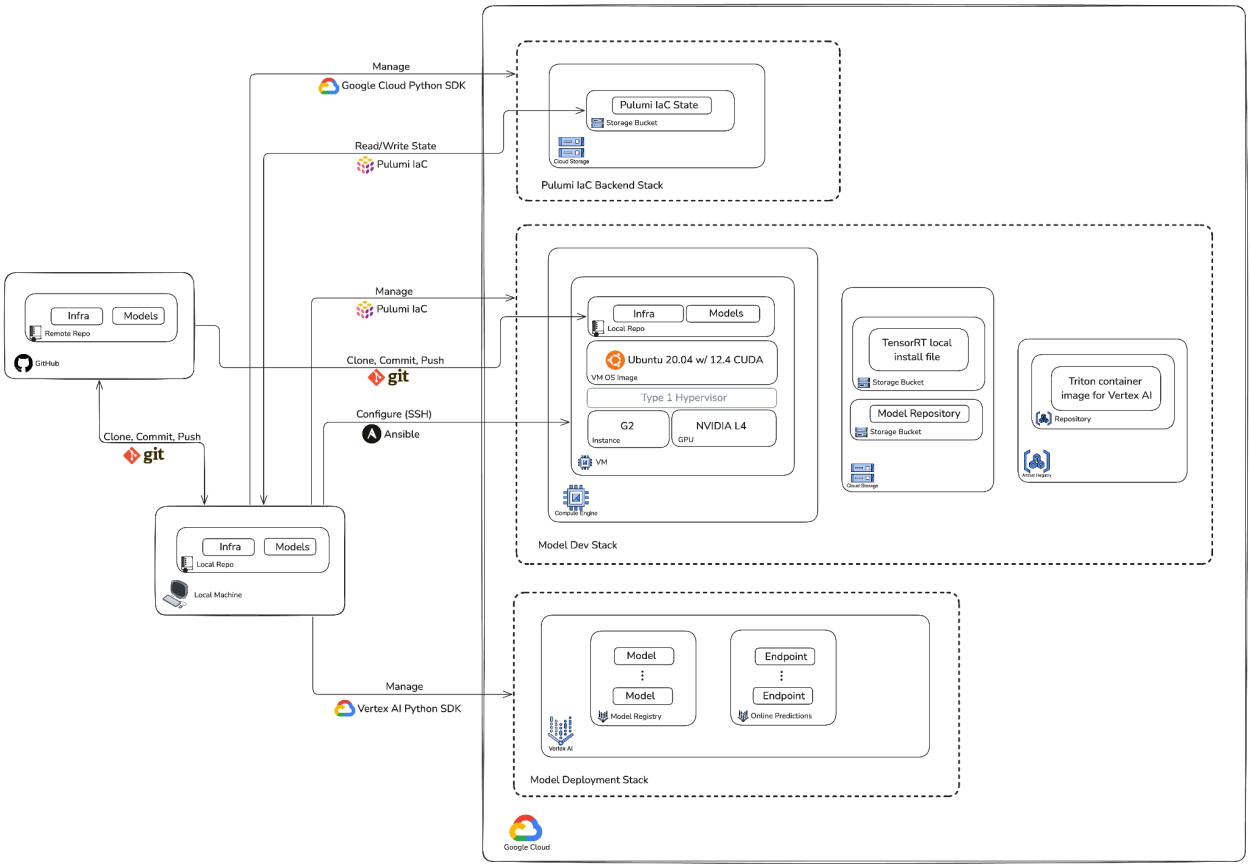

Figure 3. ML Platform Infrastructure Stacks: (top) Pulumi IaC Backend, (middle) Model Dev, (bottom) Model Deployment

ML Deploy

For deployment, we used Vertex AI's managed prediction endpoints, which allowed us to serve optimized models via custom Docker containers running NVIDIA Triton Inference Server.

We deployed each model: baseline, pruned, sparse, and quantized, to its own dedicated endpoint on Google Cloud Vertex AI. To do so with our GPU inference quota limit, we selected different Google Cloud regions for each endpoint. Each deployment ran on its own NVIDIA L4 GPU-enabled node, the same type of node as used for model development since TensorRT optimizes for a specific GPU architecture.

Model Development

We started with a pretrained ResNet50 image classification model taken from PyTorch and converted it to the ONNX format for intake by TensorRT. From there, we applied a series of post-training optimization techniques such as pruning, sparsification, and quantization using NVIDIA TensorRT Model Optimizer on our GPU-enabled VM. Each technique was applied independently to create three optimized variants of the base model. After optimization, we built the TensorRT engine offline and saved the L4 optimized models, aka TensorRT engines, in our model repository, ready for deployment to individual Vertex AI endpoints.

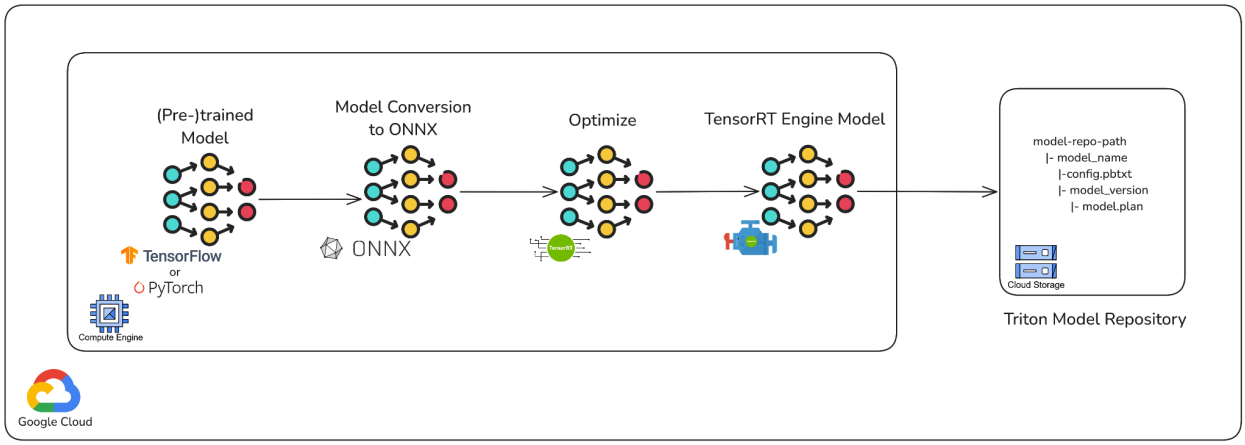

Figure 4. Model Optimization Overall Process

Optimization Techniques

- **Pruning:** Targeted subnetworks that preserved 70% of base ResNet50's FLOPs through binary search over architectural hyperparameters, maximizing accuracy under compute constraints1.
- **Sparsification:** Implemented SparseGPT's data-driven approach rather than the magnitude pruning or random approaches.
- **Quantization:** Used 1,000 ImageNet images for calibration, following research showing this size sufficient for static quantization accuracy. Applied per-channel quantization with layer fusion (Conv+BN+ReLU) to minimize error propagation.

The model sizes after the optimization techniques can be seen in the next table, and as expected they are smaller compared to the un-optimized model.

ResNet50	ResNet50 Pruned	ResNet50 Sparse	ResNet50 Quantized
128.9MB	52.4MB	128.4MB	34.5MB

We wanted to use TensorRT for our project, however, there are alternative tools that can perform similar optimizations. Here's a table of the most notable alternatives that support the same optimizations we performed:

Tool/Framework	Notes
TensorFlow Model Optimization Toolkit (TF MOT)	● Provides APIs and tools for pruning, quantization, clustering, and distillation during or after training in TensorFlow.
OpenVINO	● Intel's toolkit for optimizing models for CPUs, GPUs, VPUs; strong quantization support.
Apache TVM	● Hardware-agnostic, supports many backends; flexible with custom passes for pruning/quantization.
ONNX Runtime (with Olive Toolkit)	● Olive toolkit enables quantization and other optimizations across CPUs, NPUs, and NVIDIA GPUs.

Web App

We built a custom web app using Streamlit to make it easy for users to interact with and benchmark all four deployed endpoints simultaneously. The app allows users to upload an image, which is then sent out as inference requests to the baseline, pruned, sparse, and quantized model endpoints. To ensure a fair and efficient comparison, our app uses multithreading to send requests to all four endpoints at the same time. Each thread handles the request and response cycle for a single endpoint, allowing us to capture inference results and response times in parallel rather than sequentially. The endpoints simply return an index, referring to a class within the ImageNet1000 dataset which we then map to in order to get the

actual class name of the prediction. After all threads complete, the app displays the classification predictions, confidence scores, and response times for each endpoint together.

Whole Picture

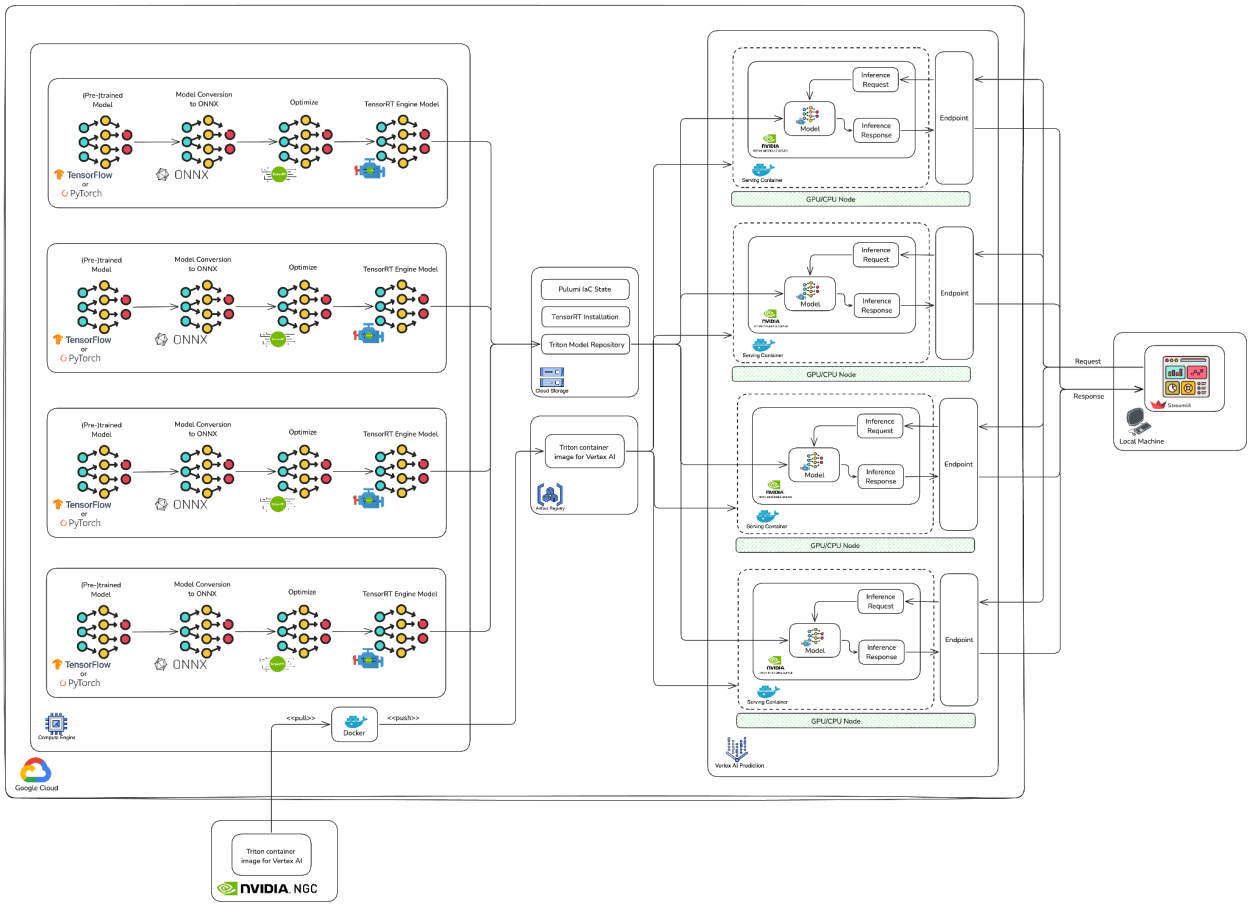

Figure 5. End-to-end Implementation for 4 ResNet50 Models

Evaluation

Figure 6 shows the result of sending a frog image to all four endpoints with our web app at the same time. Please note that the base model only accepted batches of 32 images, and we couldn't figure out in time how to reduce it to batches of 1 image in time. The y-axis on the graph is seconds.

The results show that post-training optimization techniques like pruning, sparsification, and quantization can reduce inference latency for cloud-based image classification with some accuracy loss.

Model Comparison Summary

	Model	Top Prediction	Confidence	Response Time (s)	Location	Batch Size
0	Pruned	jackfruit, jak, jack	3.1431	2.326	us-central1	1
1	Sparse	sulphur-crested cockatoo, Kakatoe galerita, Cacatua galerita	6.3566	2.288	us-east4	1
2	Quantized	tailed frog, bell toad, ribbed toad, tailed toad, Ascaphus trui	6.1126	2.368	us-west1	1
3	Base	tree frog, tree-frog	15.6493	10.503	us-east1	32

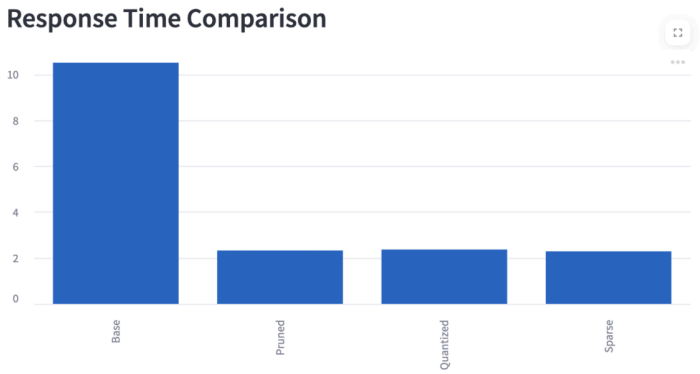

Figure 6. Frog Image Inference Results

Discussion

We ran into issues with the TensorRT installation and versioning, long Vertex AI deployment times, and some manual deployment steps.

Nevertheless, we learned how managed ML platforms like Vertex AI and Triton simplify deployment but introduce their own complexities. Also, how optimization techniques can yield substantial real-world latency improvements at the cost of accuracy.

Conclusion & Future Work

In conclusion, our project showed that post-training optimization techniques like pruning, sparsification, and quantization can reduce inference latency for cloud-based image classification with some accuracy loss. For future work, we would expand our benchmarking with larger datasets and load testing, automate our deployment pipeline, and explore multi-model endpoints with Triton Server to reduce cloud spend.

Notes on Building and Launching the Application

Due to heavy usage on GCP compute costs, we had to shutdown the deployed endpoints at the time of this submission. Please refer to the *infra/stack_deploy/README.md* for instructions on how to create a new endpoint, upload a model to the Vertex AI registry, and deploy.

References

- Homework 3 - Simple DL Workflow in Kubeflow
- Cole Bailey: Cooking up a ML Platform - Growing pains and lessons learned | PyData YouTube
- NVIDIA Triton Inference Server and its use in Netflix's Model Scoring Service | Outerbounds YouTube
- Class Presentation - NN Optimization Frameworks: Torch2Compile, TensorRT, Faster Transformer
- How to Put AI Models into Production: A Guide to Accelerated Inference | NVIDIA Book
- 2024 State of AI Inference Infrastructure Survey Report by BentoML
- Chapter 5. Building Infrastructure Stacks as Code | O'Reilly Book by Kief Morris
- How Pulumi Works | Docs Pulumi IaC Concepts
- Visual Guide to Quantization