Machine Learning Engineering & AI

Bootcamp Capstone

Step 9: Pick Your Deployment Method

17 April 2025

Deployment and Lifecycle Management of a Fine-Tuned Depth Estimation Model

"Exploring Deployment Options, Monitoring, and Integration into a Full ML Pipeline"

1. Introduction

Deploying a machine learning (ML) model in production is only one part of a broader lifecycle management process. This paper discusses the method I chose for the deployment of a fine-tuned Monocular Depth Estimation (MDE) model—as a web-based application that uses a Flask backend, a JavaScript/HTML frontend, and a Docker container—to Google Cloud Run. In addition to discussing deployment mechanics, this paper examines ongoing model care (monitoring, logging, retraining, and redeployment) and compares alternative deployment platforms. I also explain how the deployment phase fits into the complete ML pipeline involving data processing, model training, and fine-tuning.

2. Deployment Options Overview

2.1 Google Cloud Run (chosen method)

Google Cloud Run offers a fully managed containerized service with a pay-for-what-you-use pricing model. Its advantages include:

- Scalability: Automatically scales to zero when idle and scales up during bursts of traffic.
- Maintenance-Free: Google manages infrastructure, security, and scaling.
- Integration: Seamlessly integrates with other Google Cloud services such as Container Registry, Cloud Monitoring, and Cloud Logging.

Deployment on Google Cloud Run involves:

- Containerizing the Flask backend and JavaScript frontend via a Dockerfile.
- Pushing the Docker image to Google Container Registry (or Artifact Registry).
- Deploying the image to Cloud Run where it serves HTTP requests.
- Using environment variables (e.g., the PORT variable) to allow dynamic port binding.

2.2 Dedicated Virtual Machines (VMs)

Using VMs on platforms like Google Compute Engine (GCE) or Amazon EC2 is a more traditional option. Advantages include:

- Full Control: User has complete control over the infrastructure, including hardware specifications.
- Customization: Allows user to fine-tune resource allocations exactly to the user's needs.

Disadvantages/Limitations may be:

- Manual scaling and maintenance.
- More complex configuration and additional security management.
- Typically, higher costs if the instance runs 24/7 compared to serverless approaches.

2.3 Platform-as-a-Service (PaaS) Options (e.g., Heroku, AWS Elastic Beanstalk)
PaaS options offer simplified deployment and scaling without managing raw infrastructure. Advantages include:

- Ease of Use: Streamlined deployment processes.
- Managed Environment: Automatic scaling and basic monitoring.

Disadvantages/Limitations may be:

- Less fine-grained control over resource allocation.
- In some cases, higher costs for scale or slower performance compared to containerized services.

2.4 Container Orchestration Systems (Kubernetes)
Kubernetes (K8s) allows the user to manage containerized applications with complex orchestration and scaling policies. Advantages include:

- Flexibility: Highly configurable; supports multi-cloud and hybrid deployments.
- Ecosystem: Extensive support for monitoring, logging, rolling updates, and automated scaling.

Disadvantages/Limitations may be:

- Complexity: Requires significant setup and maintenance compared to fully managed services like Cloud Run.
- Operational Overhead: More technical expertise is needed to design and manage a cluster.

3. Lifecycle Management: Monitoring, Logging, and Redeployment

3.1 Post-Deployment Monitoring and Logging

Once the model is deployed, continuous monitoring is critical for:

- Performance Metrics: CPU, memory, latency, and error rates. Google Cloud Run automatically collects these metrics which can be visualized using Cloud Monitoring.
- Model-Specific Metrics: Custom metrics such as inference time, output distributions (e.g., depth range), and error metrics can be monitored via logging frameworks integrated with Flask.
- Health Checks: Automated health checks and logging help detect if the service is being terminated due to resource exhaustion (e.g., memory limits) or if there are any issues during inference.

3.2 Automated Retraining and Redeployment

Over time, the performance of ML models may degrade due to changing data distributions (concept drift) or new data emerging. A robust lifecycle management plan should include:

- Data Collection: Continuously gather new data through continuous critiquing of the deployed model.
- Model Fine-Tuning: Periodically fine-tune the MDE model using new data; fine-tuning can improve performance.
- Automated Redeployment: Establish a CI/CD pipeline that rebuilds the Docker image with the newly retrained model, pushes it to Container Registry, and deploys a new Cloud Run revision.
- Versioning and Rollback: Maintain version history so a user can revert to a previous version if the new model performs poorly.

3.3 Integration with the Full ML Pipeline

Deployment is the final stage in the complete ML lifecycle. My pipeline for the fine-tuned Depth Anything v2 MDE model includes:

- Data Processing/Data Augmentation: Preprocessing steps that augment images with features to incorporate desirable patterns to learn during model fine-tuning and inference.
- Model Training and Fine-Tuning: Model fine-tuning to adapt the model to specific datasets (e.g., mirror, indoor art, outdoor art).
- Evaluation: Use validation and test splits to evaluate the performance of the model.
- Deployment: Containerizing and deploying the model for production inference.
- Feedback Loop: Monitoring performance, capturing user feedback, and logging errors provide inputs for further model improvements.

The deployment process must align with these stages, ensuring that new model versions integrate seamlessly with the backend inference service and that monitoring tools trigger alerts when model performance changes.

4. Cost, Speed, and Performance Considerations

4.1 Cost

Cloud Run Advantages:

Pay per request and only for the resources used, which is cost-effective for variable or low traffic.

Cost Trade-offs:

Fully managed services can sometimes be more expensive at high scale compared to reserved instances on VMs, but allow a user to save on operational overhead.

4.2 Speed of Deployment

Cloud Run and PaaS:

These services offer rapid deployment with minimal configuration, suitable for fast iterations.

Self-Hosting or Kubernetes:

These require more setup and configuration, potentially delaying the deployment process.

4.3 Model Inference Performance

Cloud Run:

Provides auto-scaling and load balancing; however, cold starts can add latency. Optimizing a container (e.g., using pre-warmed instances) can mitigate this.

Dedicated Hosting:

May reduce cold start times, but introduces additional management overhead and scaling challenges.

4.4 Monitoring Options and Logging

Cloud Monitoring Integration:

Google Cloud Run automatically integrates with Cloud Monitoring and Cloud Logging.

Custom Logging:

Integrating a logging system within my Flask application (e.g., using Python's logging module or structured logs) can provide more detailed insights into model performance and errors.

5. Summary and Recommendation

Deploying my fine-tuned Depth Anything v2 MDE model as a web application required careful consideration of both the technical deployment strategy and the ongoing lifecycle management. I considered different options including Google Cloud Run which offers a managed, scalable, and serverless platform that minimizes operational overhead. I also considered alternatives like self-hosting on VMs, PaaS, or Kubernetes which provide more control however may require a higher level of maintenance. To decide on the deployment platform, I weighed the following considerations:

- Monitoring and Maintenance:

  Integrated monitoring, logging, and automated retraining/redeployment provide a robust lifecycle management plan.
- Cost and Performance:

  Cloud Run is cost-efficient for variable loads but may incur higher costs at scale if not optimized.
- Cold start issues can be mitigated by pre-warming instances or optimizing the container configuration.

Recommendation:

For my project with the end-goal of sharing research and collaborating, and involving variable workloads and a desire for minimal maintenance, I decided Google Cloud Run is an attractive option from an efficiency and cost standpoints. Google Cloud Run offers seamless integration into a modern CI/CD pipeline with Cloud Build, automated scaling, and built-in monitoring/logging. The deployment process seems straightforward and simple from a user's standpoint – a streamlined process requiring the user to combine the Flask backend and JavaScript frontend into a container with Docker, and then tagging, pushing, and running the container on Cloud Run. Additionally, for ongoing work if desired, an automated feedback loop for retraining may be established to ensure the deployed model remains robust over time.