Full-stack Machine Learning

Alexander D’hoore

Full-stack Machine Learning

- Introduction to MLOps
- Comparison to DevOps

- Experiment Tracking
- **Lab 1**

- Data Engineering
- **Lab 2**

- Model Deployment
  - => **you are here**
  - Cloud, edge
  - Prototyping
- **Lab 3**

- Automation
  - Agents, pipelines
  - Hyperparameters
- **Lab 4**

Model Deployment

Introduction to Model Deployment

- Model deployment refers to the process of integrating a machine learning model into an existing **production environment** to make predictions based on new data.

- It ensures that the model is accessible to users or other systems for **inference**.

- Deployment is a critical step to realize the **practical value** of a machine learning model.

Importance of Deployment

- **Bridges the gap** between model development and real-world application.
- Enables **automation and scalability** of model predictions.
- Facilitates continuous **improvement and updates** of models based on new data.

Types of Deployment: Cloud vs Edge

- **Cloud Deployment**: Models are deployed on **centralized servers** accessible via the internet.

- **Edge Deployment**: Models are deployed on **local devices**, closer to where the data is generated.

Cloud Deployment

Cloud Deployment: Benefits

- Cloud deployment involves hosting models on cloud servers, allowing for remote access via **APIs**.

- **Benefits**:
  - **Scalability**: Easily handle varying loads by scaling resources up or down.
  - **Accessibility**: Models can be accessed from anywhere via the internet.
  - **Maintenance**: Simplifies updates and management of models.
  - **Cost Efficiency**: Pay-as-you-go pricing models and resource sharing.

Cloud Deployment: Use Cases

- **E-commerce**: Product recommendation systems.
- **Healthcare**: Predictive analytics for patient data.
- **Finance**: Fraud detection and risk assessment.
- **Retail**: Demand forecasting and inventory management.

REST APIs in Cloud Deployment

- **What are REST APIs?**

- Representational State Transfer (REST) APIs
- allow systems to communicate over **HTTP**
- using standard methods like **GET, POST, PUT, and DELETE**

- They enable interaction with machine learning models deployed on cloud servers.

Role of REST APIs

- Provide a **standardized** way to expose model prediction functionality.
- Allow different applications to request and receive **model predictions**.
- Facilitate **integration** with web and mobile applications.

- **Example Workflow**: Model as a REST API
    1. Develop and train the machine learning model.
    2. Deploy the model on a cloud server.
    3. Wrap the model inference logic in a REST API.
    4. Clients send data to the API endpoint and receive predictions in response.

Cloud Providers for Model Deployment

- Overview of Popular Cloud Providers:

- AWS (Amazon Web Services)
- GCP (Google Cloud Platform)
- Azure (Microsoft Azure)
- IBM Cloud
- Oracle Cloud

Cloud Providers: Features

- General Features Offered by Cloud Providers:

- **Compute Resources**: Virtual machines, serverless functions, and container services.
- **Storage Solutions**: Databases, object storage, and data lakes.
- **Networking**: Load balancing, virtual private clouds, and content delivery networks.
- **Security**: Identity and access management, encryption, and compliance certifications.

Model Serving Engines

- Model **serving engines** are specialized platforms designed to deploy and manage machine learning models at **scale**.
- They streamline the process of serving models and handling requests.

- Key Features to Look For:
  - **Scalability**: Ability to handle high request volumes.
  - **Latency**: Low response times for real-time predictions.
  - **Monitoring**: Tools to track model performance and usage.
  - **Integration**: Compatibility with machine learning frameworks.
  - **Management**: Version control, rollbacks, and updates.

Model Serving Engines: Examples

- Examples of Model Serving Engines

- TensorFlow Serving
- TorchServe
- KFServing (KServe)
- BentoML
- Seldon Core

Case Study: Cloud Deployment

- A supermarket wants to deploy a demand forecasting model to optimize inventory management.

- Deployment Process
  - **Data Preparation**: Clean and preprocess the historical sales data.
  - **Model Training**: Develop and train the demand forecasting model.
  - **Model Deployment**: Deploy the model to a cloud server using a serving engine.
  - **API Setup**: Create a REST API endpoint for the model.
  - **Monitoring**: Implement monitoring and logging to track model performance.

Edge Deployment

Introduction to Edge Deployment

- Edge deployment refers to running machine learning models on **local devices** rather than in the cloud.

- Benefits:
  - **Low Latency**: Faster inference times by processing data locally.
  - **Reduced Bandwidth**: Less data transmitted over the network.
  - **Privacy and Security**: Sensitive data stays on local devices.
  - **Offline Capability**: Models can function without internet connectivity.

Possible Edge Devices

- **Mobile Devices**: Smartphones and tablets.
- **IoT Devices**: Sensors, smart home devices, and wearables.
- **Embedded Systems**: Industrial controllers, automotive systems.
- **Edge Servers**: Local servers with limited compute resources.
- **Single-Board Computers**: Raspberry Pi, NVIDIA Jetson.

TensorFlow Lite

- TensorFlow Lite is a **lightweight solution** for deploying TensorFlow models on mobile and embedded devices.
- **Key Features**
  - **Model Conversion**: Convert TensorFlow models to a compact format.
  - **Interpreter**: Execute models on-device with minimal footprint.
  - **Delegates**: Hardware acceleration using GPU, DSP, or other specialized processors.
- **Use Cases**
  - Real-time image classification on mobile phones.
  - Speech recognition in embedded systems.
  - Object detection on IoT devices.

ONNX (Open Neural Network Exchange) Format

- ONNX is an **open standard** for representing machine learning models.
  - Enables interoperability between different frameworks.
- **Key Features**
  - **Model Interoperability**: Transfer models between frameworks like PyTorch, TensorFlow, and others.
  - **Optimization**: Tools to optimize models for performance on various hardware.
- **Use Cases**
  - Converting models developed in PyTorch to run on different platforms.
  - Sharing models across teams using different machine learning frameworks.

Microsoft ONNX Runtime

- ONNX Runtime is a cross-platform, **high-performance inference engine**.
  - For Open Neural Network Exchange (ONNX) models.
- **Key Features**
  - **Cross-Platform**: Runs on Windows, Linux, Mac, and various edge devices.
  - **Optimizations**: Hardware-specific optimizations for CPUs, GPUs, and other accelerators.
  - **Extensibility**: Custom operators and execution providers.
- **Use Cases**
  - Deploying complex models on powerful edge device.
  - Integrating ONNX models into existing applications for real-time inference.

NVIDIA TensorRT

- TensorRT is a **high-performance** deep learning inference optimizer and runtime library developed by NVIDIA.
- **Key Features**
  - **Model Optimization**: Converts and optimizes models for NVIDIA GPUs.
  - **Low Latency**: Minimizes inference time with advanced optimizations.
  - **Precision Calibration**: Supports FP16 and INT8 precision for faster performance.
- **Use Cases**
  - Deploying deep learning models on NVIDIA Jetson for robotics.
  - Accelerating AI applications in autonomous vehicles.
  - Real-time video analytics on edge devices.

Best Practices for Edge Deployment

- Model Optimization
  - **Quantization**: Reduce model size by converting weights to lower precision (e.g., INT8).
  - **Pruning**: Remove redundant neurons and weights.
  - **Compression**: Use techniques like weight clustering and Huffman coding.

- Resource Management
  - **Memory**: Optimize memory usage to fit models on limited hardware.
  - **Compute**: Balance model complexity with available processing power.
  - **Energy**: Minimize energy consumption for battery-powered devices.

Case Study: Edge Deployment

- A smart home company wants to deploy a voice recognition model on their IoT devices to enable voice commands.

- Deployment Process
  - **Data Preparation**: Collect and preprocess audio data for training.
  - **Model Training**: Develop and train the voice recognition model.
  - **Model Conversion**: Convert the model to TensorFlow Lite format.
  - **Device Deployment**: Deploy the model on IoT devices.
  - **Integration**: Integrate the model with the device firmware to handle voice commands.

Summary: Cloud vs Edge Deployment

- Cloud Deployment
  - **Scalability**: Easily handles varying loads
  - **Accessibility**: Global access via the internet
  - **Maintenance**: Simplified updates
  - **Challenges**: Higher latency, bandwidth costs, data privacy
- Edge Deployment
  - **Low Latency**: Fast local processing
  - **Privacy**: Data stays on device
  - **Offline Capability**: Works without internet
  - **Challenges**: Limited resources, harder maintenance

Prototyping Apps

Introduction to Prototyping Apps

- Prototyping apps allow **rapid development** and deployment of interactive machine learning demos and applications.

- Use Cases:
  - **Internal Demos**: Quickly create interactive apps to showcase models and data insights.
  - **Custom Labeling**: Develop labeling tools for data annotation tasks.
  - **Monitoring Dashboards**: Build comprehensive dashboards to track metrics and visualize data.

Prototyping Apps: Examples

- **Image Classification**: Upload an image and get predictions.
- **Text Analysis**: Input text and receive sentiment analysis results.
- **Audio Processing**: Record or upload audio and get transcription.
- **Data Visualization**: Create interactive plots and charts.
- **Model Exploration**: Adjust model parameters and observe changes in output.
- **Data Analysis**: Perform exploratory data analysis with interactive controls.

Prototyping: Gradio

- Gradio is an open-source library for creating **interactive user interfaces** for machine learning models.
- Simplifies sharing models with non-technical stakeholders.

- Key Features:
  - **Easy Integration**: Quickly wrap models and functions with a web interface.
  - **Interactive Demos**: Create live demos for model inference.
  - **Customizable Interfaces**: Build interfaces with buttons, sliders, text inputs, and more.
  - **Deployment Options**: Host locally or deploy on the web with minimal effort.

Prototyping: Streamlit

- Streamlit is an open-source app framework for creating and sharing **data applications**.
- Focuses on simplicity and speed for building **custom web applications**.

- Key Features:
  - **Python-Based**: Build apps using pure Python code.
  - **Real-Time Updates**: Automatically update the app as code changes.
  - **Widgets and Controls**: Use sliders, dropdowns, and other UI elements to interact with data.
  - **Deployment Options**: Deploy locally or use Streamlit Cloud for online hosting.

Comparison: Gradio vs Streamlit

- **Gradio**
  - Best for quick, interactive model demos.
  - Simple to create interfaces for machine learning models.
  - Ideal for non-technical stakeholders to interact with models.

- **Streamlit**
  - Best for building comprehensive data applications and dashboards.
  - Greater flexibility and control over the app layout and functionality.
  - Suitable for interactive data visualization and analysis.

Lab 3: Deployment

Lab 3: Deployment

- We will learn how to build **Gradio apps**

- Follow the tutorial at:

https://www.gradio.app/guides/quickstart