DEPLOYING MACHINE LEARNING MODELS

WORKSHOP DETAILS

- Instructor: Tanya Khanna
- Email: tk759@scarletmail.rutgers.edu
- Course Materials: Github Link - https://github.com/Tanya-Khanna/DataScienceWorkshop_Fall-2024_NBL
- Workshop Recordings: https://libguides.rutgers.edu/datascience/python
- Spring 2024 Workshops: Link
- Workshop Feedback Form: https://rutgers.libwizard.com/f/graduate_specialist_feedback

WORKSHOPS SCHEDULE

Introduction to Python Programming	September 9, 2024; 4 - 5:30 PM
Advanced Python Programming	September 16, 2024; 4 - 5:30 PM
Web Scraping with Python	September 23, 2024; 4 - 5:30 PM
Mastering Data Analysis: Pandas and Numpy	September 30, 2024; 4 - 5:30 PM
Data Management with Python: SQL and NoSQL	October 7, 2024; 4 - 5:30 PM
Python for Visualization and Exploration	October 14, 2024; 4 - 5:30 PM
Introduction to Machine Learning: Supervised Learning	October 21, 2024; 4 - 5:30 PM
Introduction to Machine Learning: Unsupervised Learning	October 28, 2024; 4 - 5:30 PM
Deploying Machine Learning Models	November 4, 2024; 4 - 5:30 PM
Ethical AI and Responsible Data Science	November 11, 2024; 4 - 5:30 PM
Large Language Models (LLMs) and ChatGPT	November 18, 2024; 4 - 5:30 PM

https://libcal.rutgers.edu/calendar/nblworkshops?cid=4537&t=d&d=0000-00-00&cal=4537&inc=0

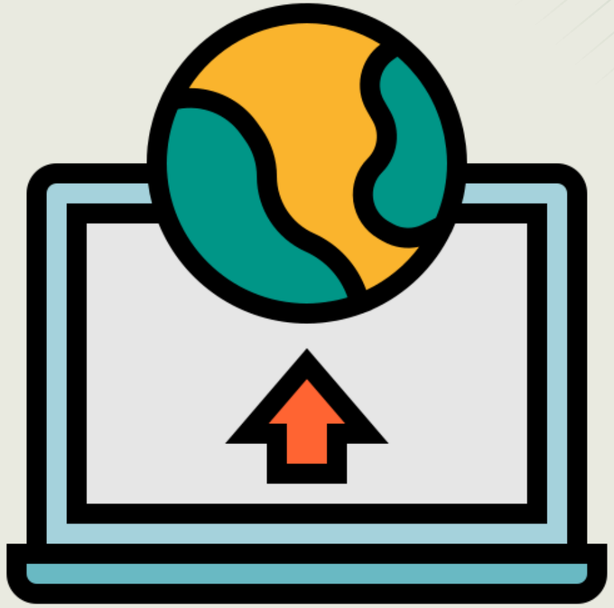
WORKSHOP AGENDA

- What is Model Deployment?
- Types of Model Deployment
- CI/CD Pipeline for ML
- Model Monitoring and Logging
- Best Practices for Deployment
- Practical Session: Deploying a Machine Learning Model (Using FastAPI and Streamlit)

INTRODUCTION TO MODEL DEPLOYMENT

What is Model Deployment?

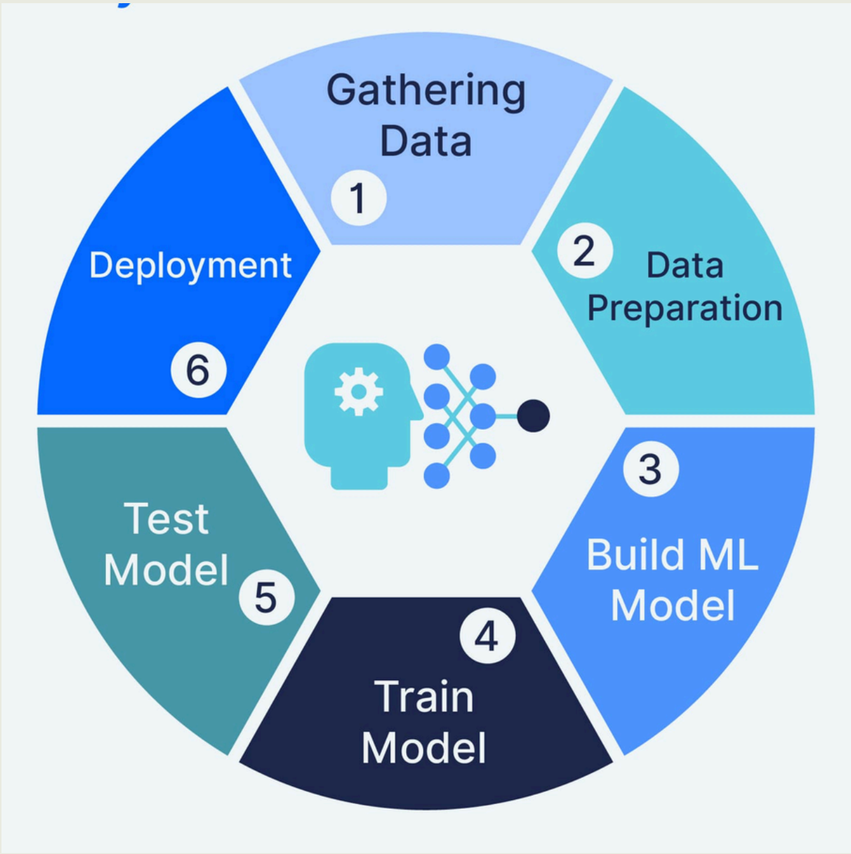
Think of it as moving a dish from the kitchen to the dining table—training is cooking, but deployment is when it’s ready to be served and enjoyed. Thus, it is the process of making a trained ML model accessible to users or systems to make predictions in real time.



Importance of Model Deployment

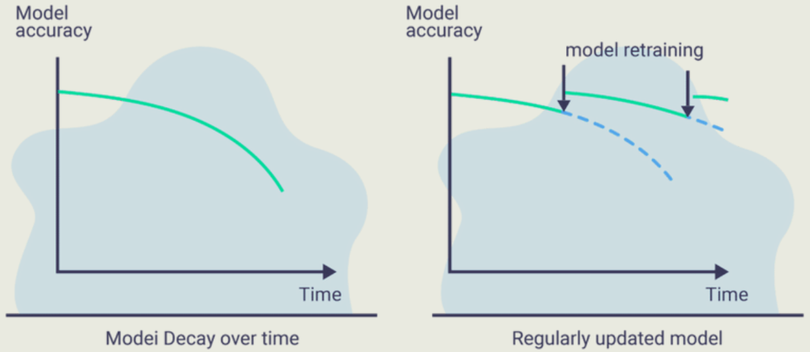
- **Turning Ideas into Impact:** Without deployment, an ML model is like a "prototype" that only lives on a researcher’s computer. Deployment allows it to impact real-world applications, like suggesting products in e-commerce or predicting loan eligibility in finance.
- **Continuous Learning:** Deployment is also important for gathering new data, which can help models improve over time.

MACHINE LEARNING LIFECYCLE



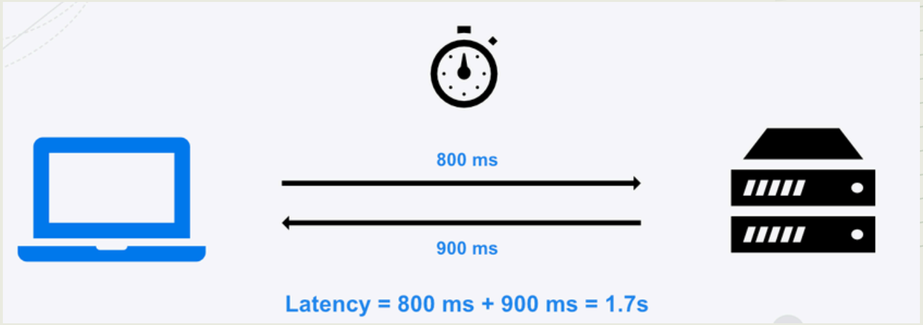
- **Step 1: Data Processing**
  - Preparing data in a way that the model can "understand." This could mean cleaning data, converting formats, or normalizing values. Garbage in, garbage out—a model is only as good as the data it's given.
- **Step 2: Model Training**
  - Training is where the model learns from historical data. Think of it as teaching a student using past exam papers. Training a model well is critical, but it's just the starting point. Without deployment, training alone has no real-world impact.
- **Step 3: Evaluation**
  - Testing the model on new data to ensure it performs well. Evaluation is like a final exam, checking if the model has truly "learned" and can generalize beyond its training data.
- **Step 4: Deployment**
  - This is where we actually put the model in action. It could mean integrating it into an app, website, or a system where it can start making predictions. Think of deployment as ongoing—monitoring and improving deployed models is essential for them to stay useful over time.

CHALLENGES IN DEPLOYMENT



Model Drift

Over time, the data a model was trained on may no longer represent real-world data. For example, a model predicting housing prices trained in 2020 may no longer be accurate in 2024. Detecting and adapting to this drift is crucial to ensure predictions stay accurate.



Latency

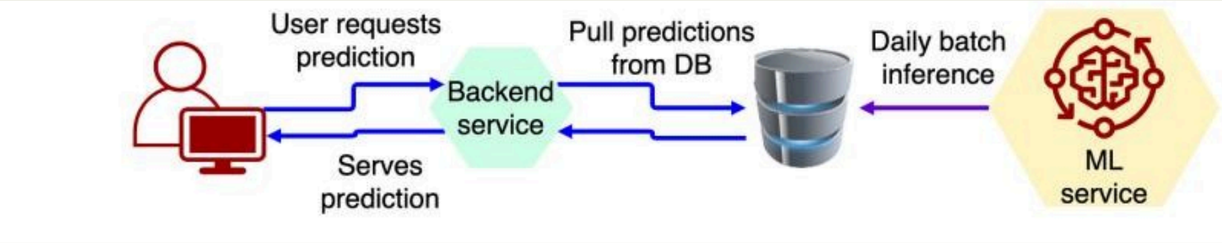
Latency is the time it takes for a model to make a prediction after receiving data. Imagine asking a voice assistant a question and waiting minutes for a response—low latency ensures quick results. Achieving low latency is important, especially in high-stakes fields like healthcare or finance.

Scalability

Scalability is the model's ability to handle increasing numbers of requests or large datasets without degrading performance. Think about a social media platform during a viral event—models need to handle the spike in demand smoothly.

TYPES OF DEPLOYMENT

Batch deployment



- User Interaction: The user sends a request for a prediction to the backend service.
- Backend Service: This service manages requests and serves the stored predictions.
- Database (DB): The backend pulls predictions from the database where the predictions are stored.
- ML Service: Predictions are generated by the ML model through daily batch inference, meaning the model processes data in batches (often at scheduled times, like once a day) and updates predictions in the database.

Batch deployment generates predictions at set times (like daily or weekly) instead of instantly. It works well when data doesn't change quickly, or immediate predictions aren't needed. Rather than processing each request in real time, predictions are made in groups (batches) and stored in a database for future use. This approach is ideal for consistent, non-urgent predictions where a delay is acceptable.

Framework/Tools for Batch Processing

- **Apache Spark:** Handles large datasets with distributed batch processing and parallel computing.
- **Airflow:** Schedules and manages batch jobs to keep predictions up-to-date.
- **Dask:** Enables parallel processing for big datasets, perfect for batch tasks.

Pros and Cons of Batch Deployment

When to Use Batch Deployment

Cost-Effective

Suitable for scenarios where real-time predictions are unnecessary, reducing computational costs.

Consistency

Ideal for non-urgent applications where predictions can be generated at scheduled times.

Reduced Load on Models

By precomputing predictions, batch deployment reduces the load on the model, as it doesn't need to generate predictions on-demand for every user request.

Data Volume

Works well when you need to process large datasets that can be scheduled at off-peak times.

Examples: Daily sales forecasts, recommendation updates, and financial reports.

When Not to Use Batch Deployment

Not Suitable for Real-Time Applications

Not ideal for use cases that require immediate responses, like fraud detection or real-time recommendation systems.

Latency

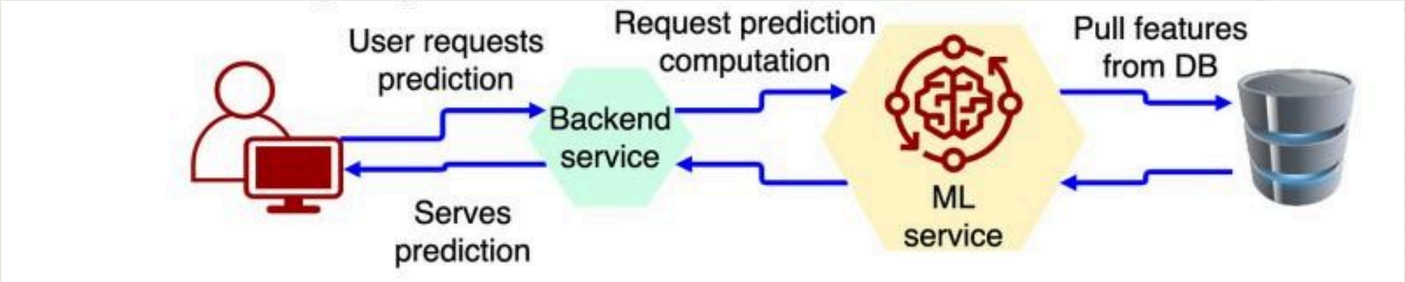
There can be a lag between data changes and the availability of updated predictions, which may affect decision-making for time-sensitive applications.

Data Staleness

Predictions may become outdated between batch runs, which is problematic in environments with rapidly changing data.

Examples: Real-time stock trading predictions, emergency response systems, or dynamic pricing.

Real-time deployment



Real-time deployment is a process where a machine learning model makes predictions instantly when a request is made. Instead of precomputing predictions, the model processes data on demand, retrieving necessary information and computing results in real time. This type of deployment is essential for applications that need immediate responses.

Framework/Tools for Batch Processing

- **TensorFlow Serving:** Delivers TensorFlow models for real-time predictions in production.
- **Docker and Kubernetes:** Manage and scale real-time applications across servers.
- **Apache Kafka:** Streams real-time data for instant processing needs.

Pros and Cons of Real-Time Deployment

When to Use Real-Time Deployment

Immediate Responses

Ideal for applications where instant predictions are critical, like fraud detection or personalized recommendations.

Up-to-date Predictions

Ensures predictions are based on the latest data, useful for dynamic environments.

Enhanced User Experience

Provides quick feedback, improving interactions with users who expect fast responses.

Examples: Chatbots, recommendation engines, and live monitoring systems.

When Not to Use Real-Time Deployment

Higher Costs

Real-time systems require significant computational resources, which can be costly.

Scalability Challenges

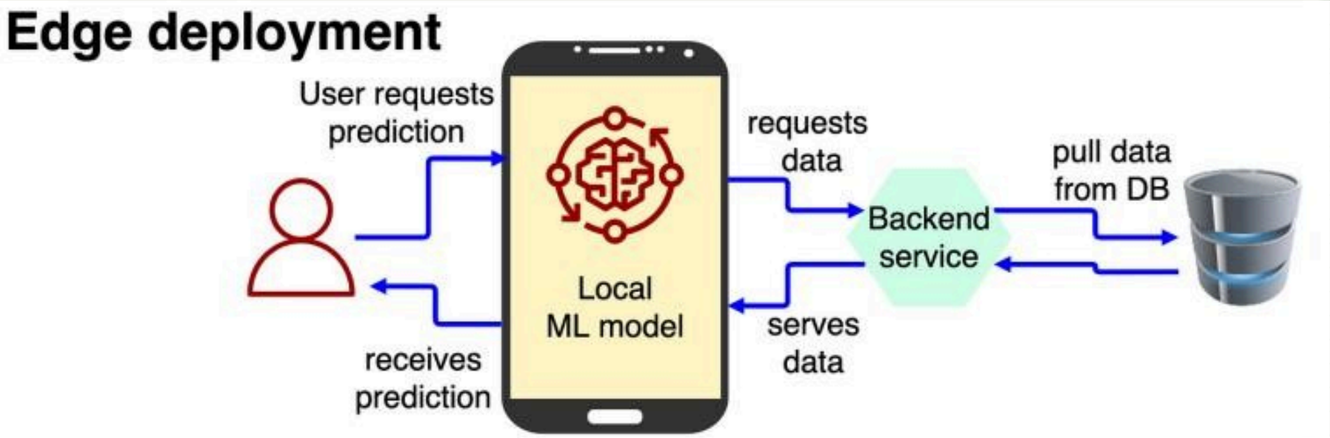
Handling large numbers of real-time requests can strain the system if not managed well.

Complexity

Real-time deployment requires more infrastructure and may need optimization for speed and latency.

Examples: Not ideal for applications where predictions don’t need to be updated frequently, like monthly sales forecasts.

Edge deployment



Edge deployment is the process of deploying machine learning models on local devices (like smartphones, IoT devices, or sensors) instead of centralized servers. This allows predictions to be made directly on the device, reducing the need to send data to the cloud for processing. Edge deployment is beneficial for scenarios where low latency, privacy, and offline functionality are critical.

Framework/Tools for Batch Processing

- **TensorFlow Lite:** A lightweight version of TensorFlow designed for mobile and embedded devices.
- **ONNX Runtime:** Optimizes and deploys ONNX models on various edge devices.
- **AWS IoT Greengrass:** Enables edge devices to act locally on data while still using cloud services when needed.

Pros and Cons of Edge Deployment

When to Use Edge Deployment

Low Latency

Processing data locally reduces response time, making it ideal for applications that require instant feedback.

Enhanced Privacy

Data stays on the device, which is beneficial for sensitive applications where privacy is a priority.

Offline Functionality

Edge deployment allows models to function without internet access, which is valuable in remote or disconnected environments.

Examples: Autonomous vehicles, wearable health devices, and smart home automation.

When Not to Use Edge Deployment

Limited Processing Power

Edge devices have limited computational resources, which can restrict the complexity of models that can be deployed.

Device Compatibility

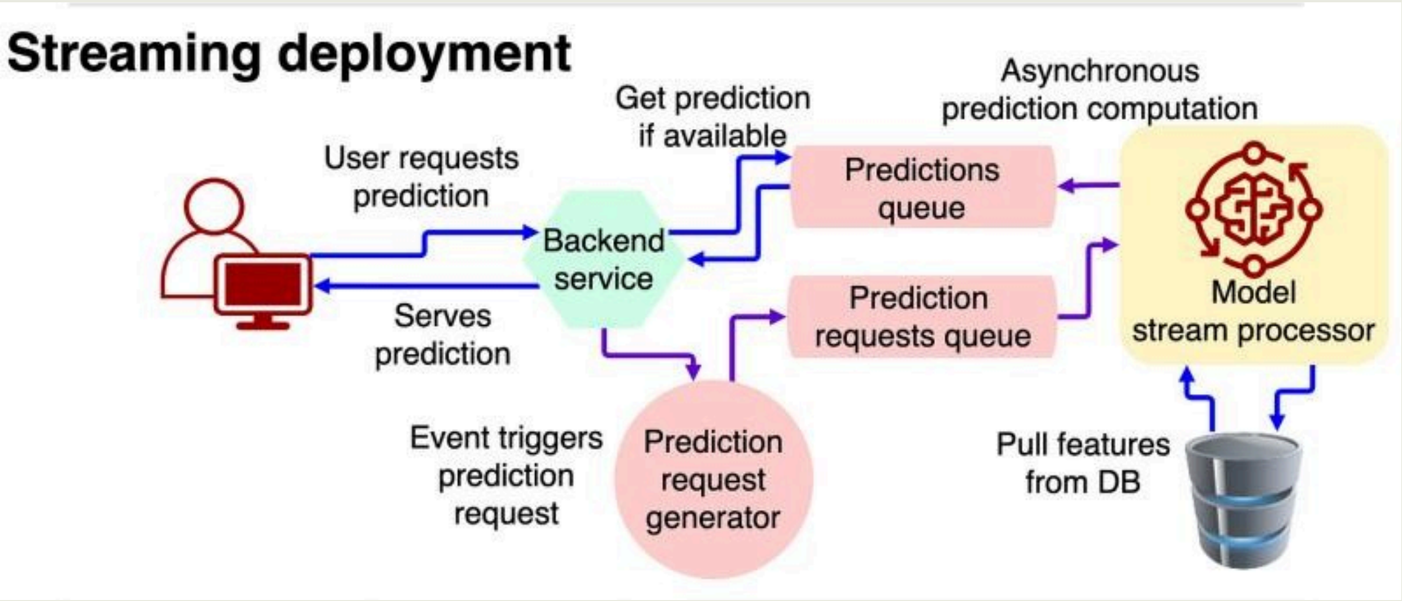
Models may need optimization to run on different types of edge devices, adding complexity.

Update Challenges

Updating models on multiple devices can be challenging compared to centralized server updates.

Examples: Not suitable for applications requiring complex models that need high computational power, such as deep neural networks for image generation.

Streaming deployment



User Interaction: The user asks for a prediction, like a recommendation or alert, from the system.
Backend Service: This service listens for the user's request and, if the prediction is ready, sends it back immediately.
Prediction Request Generator: When something happens (like new data arriving), this part of the system creates a prediction request, asking the model to generate a fresh prediction.
Predictions Queue & Prediction Requests Queue: These queues act like waiting lines, keeping prediction requests and predictions organized. One queue holds requests waiting to be processed, and another holds completed predictions ready to be sent back.
Model Stream Processor: This is where predictions are actually made. The model stream processor pulls data from the database, uses it to compute predictions, and updates the queue.
Database: Stores the data that the model needs to make predictions. The model processor retrieves this data when it needs to make a new prediction.
In short, the system works like a team: the backend serves user requests, the prediction generator triggers new predictions when needed, the queues organize requests and predictions, and the model processor does the actual prediction work using data from the database.

Streaming deployment continuously processes data in real time by handling a constant flow of data events and generating predictions as new data arrives. It's ideal for applications where predictions are needed instantly upon data arrival and where data flows continuously, such as from sensors, social media feeds, or financial transactions.

Framework/Tools for Batch Processing

- **Apache Kafka:** Handles real-time data streaming and is widely used for managing event-driven applications.
- **Apache Flink:** Processes continuous data streams and allows real-time predictions, ideal for high-throughput environments.
- **Google Cloud Dataflow / AWS Kinesis:** Cloud-based tools for real-time data processing and streaming deployment.

Pros and Cons of Streaming Deployment

When to Use Streaming Deployment

Real-Time Insights

Provides immediate predictions as soon as data arrives, valuable for applications that need instant analysis.

Event-Driven

Great for scenarios where predictions are triggered by events, such as alerts or sensor data changes.

Scalable and Efficient

Can handle high data throughput, making it suitable for applications with large, continuous data streams.

Examples: Fraud detection, live recommendation systems, and monitoring industrial equipment.

When Not to Use Streaming Deployment

High Costs and Complexity

Requires infrastructure to support continuous data processing, which can be expensive and complex.

Latency Concerns

In cases of very high data volumes, there may be processing delays if the system is not optimized.

Overkill for Static Data

Not suitable for applications with static or batch data that don’t require real-time predictions.

Examples: Monthly sales forecasting, periodic reporting, and other non-time-sensitive tasks.

INTRODUCTION TO CI/CD FOR ML MODELS

- **Continuous Integration (CI):** A process where code changes are automatically tested and integrated into the main codebase frequently. For ML, this means not only testing code but also verifying data, features, and model quality.
- **Continuous Deployment/Delivery (CD):** Automatically releasing and deploying updates once they pass all tests. In ML, CD also includes deploying updated models or data pipelines.

Importance of CI/CD in ML

CI/CD ensures that updates don't accidentally break the system and that the latest model is always performing well. It allows ML systems to scale easily by managing frequent updates to code, data, and models in a structured way. Automating integration and deployment means faster updates, which is critical for real-time applications like fraud detection or recommendations.

CI/CD PIPELINE FOR ML MODELS

PROCESS	
Code Versioning & Source Control	Just like saving versions of a document, source control lets teams save every version of their code so they can go back if something breaks. Use of tools like Git to keep track of code changes and maintain version control.
Automated Testing (Code and Data)	Automated tests ensure all parts of the ML pipeline (code, data, features) are in good shape before moving forward. Unit tests to check individual code components, data validation tests to ensure data quality, and feature tests to ensure feature engineering is consistent.
Model Training & Validation	Train the model on new data and validate performance using metrics. If the model passes quality checks, it can move to the next stage.
Model Packaging	Package the model and its dependencies (such as libraries and configurations) so it can be deployed consistently across environments.
Continuous Deployment (CD) & Monitoring	Deploy the model to production (or staging) automatically and set up monitoring to track its performance over time.

MODEL MONITORING & LOGGING

Importance of Model Monitoring

Monitoring allows us to see how well a model is performing over time. This includes tracking metrics like accuracy, precision, recall, and other key indicators of model quality. Monitoring ensures that the model continues to meet the standards it was trained to achieve. Continuous monitoring provides immediate alerts when a model's predictions start to deviate from expectations, allowing for quick intervention. This is especially important in high-stakes applications like finance, healthcare, or fraud detection.

Importance of Logging

Logging captures information about each prediction, such as input data, output predictions, and runtime errors. This data is invaluable for debugging and understanding model behavior in production. When a model doesn't perform as expected, logs provide a history of the model's operations, making it easier to trace problems back to their sources.

Model Drift: What It Is and Why It Matters

Model drift happens when a model's performance worsens because the data it's predicting on changes, making the training data less relevant. There are two types: **Data Drift:** Changes in data distribution (e.g., a new demographic using an app). **Concept Drift:** Changes in the relationship between inputs and targets (e.g., seasonal sales patterns). Drift can lead to inaccurate predictions, reducing the model's value and causing potential issues. Early detection allows teams to retrain or update the model before performance drops significantly.

Drift Detection Techniques

- Monitoring Metrics: Set thresholds for key performance metrics. A significant drop in metrics like accuracy may indicate drift.
- Statistical Tests: Use tests like Kolmogorov-Smirnov or Chi-Square to detect shifts in data distribution.
- Data Sampling: Regularly review and analyze samples of new data to check for changes in trends or patterns.

Managing Drift

- Retraining the Model: Regularly retrain the model with fresh data to keep it aligned with current trends.
- Adaptive Models: Use models that can learn continuously or adjust to changes in data over time.
- Alerts and Automation: Set up alerts to notify when drift occurs and automate retraining processes where possible.

BEST PRACTICES FOR ML MODEL DEPLOYMENT

VERSIONING

Why Versioning Matters:

- Track Changes: Allows tracking of updates to models, data, and code, making it easier to understand what changed and why.
- Reproducibility: Ensures that a specific version of a model can be reproduced, which is critical for debugging and compliance.
- Rollbacks: In case of issues, versioning allows rolling back to a previous, stable model version.

Best Practices for Versioning:

- Model Versioning: Use semantic versioning (e.g., v1.0, v1.1) for models to easily identify updates and bug fixes.
- Data Versioning: Track the version of the data used to train each model, as changes in data can affect model performance.
- Automated Versioning: Use tools like DVC (Data Version Control) or MLflow for automated version control of both data and models.

BEST PRACTICES FOR ML MODEL DEPLOYMENT

SCALABILITY TIPS

Horizontal vs. Vertical Scaling:

- Horizontal Scaling: Adding more instances of the model server (e.g., adding more servers) to handle higher loads.
- Vertical Scaling: Increasing resources (CPU, GPU, memory) on a single server to boost capacity.
- Best Practice: Horizontal scaling is often preferred for scalability because it allows the system to adapt to changing loads without requiring a single powerful machine.
- Load Balancing: Distribute incoming requests across multiple model instances to prevent overloading a single server.

Model Optimization:

- Quantization: Reducing model precision (e.g., from float32 to int8) can speed up inference with minimal impact on accuracy.
- Distillation: Use a smaller “student” model that approximates a larger, more complex “teacher” model, improving inference speed.
- Batching Requests: Processing multiple requests together can reduce computation time per prediction.

Thank you.

Now let's move onto the practical session!