Module 6 - Tuần 3 - MLOps Monitoring với Prometheus & Grafana

Time-Series Team

Ngày 21 tháng 11 năm 2025

1 Mở đầu: Khi mô hình bước vào thế giới thật

Trong môi trường nghiên cứu, mô hình Machine Learning (ML) thường hoạt động rất tốt: dữ liệu sạch, phân phối ổn định, không bị nhiễu, tài nguyên tính toán dồi dào. Tuy nhiên, khi mô hình được *deploy* vào môi trường thực tế, mọi giả định này bắt đầu "vỡ vụn":

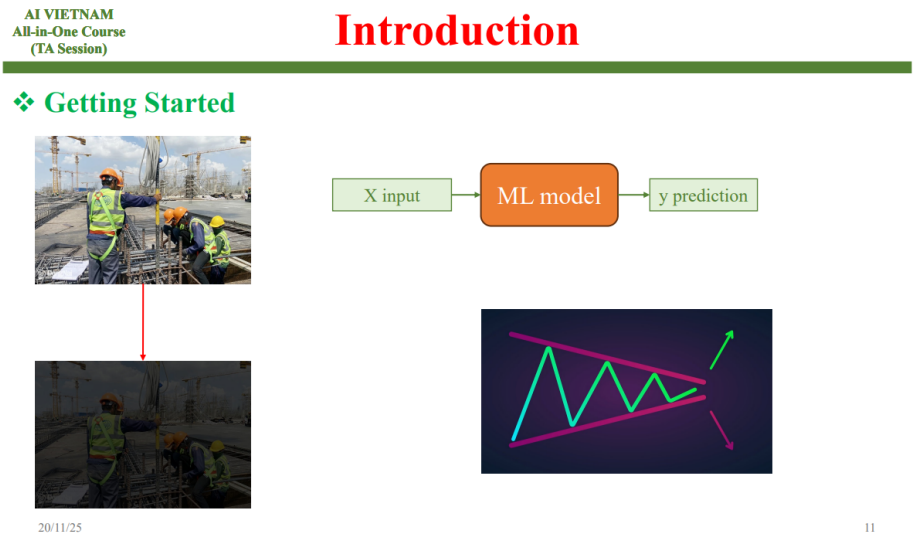
- Latency tăng bất thường, người dùng phải chờ lâu.
- Tài nguyên CPU, GPU, RAM bị *bóp nghẹt* khi có nhiều request.
- Dữ liệu đầu vào (ảnh, text, tín hiệu) thay đổi theo thời gian ⇒ **data drift**.
- Mối quan hệ giữa đầu vào và đầu ra thay đổi ⇒ **concept drift**.
- Hệ thống thỉnh thoảng trả về lỗi, nhưng không ai biết *tại sao*.

Nếu không có **monitoring** và **logging**, ta gần như "đi đêm mà không có đèn": không biết mô hình đang hoạt động ra sao, không phát hiện được sự suy giảm hiệu năng, cũng không thể giải thích được những lỗi bất thường.

Một ví dụ rất đời thực có thể thấy rõ trong các hệ thống thị giác máy tính 1. Vào ban ngày, khi ánh sáng đầy đủ và hình ảnh sắc nét, mô hình YOLO11 hoạt động gần như hoàn hảo: nó nhận diện đúng người, đúng vật thể, và đưa ra bounding box chính xác. Tuy nhiên, chỉ cần điều kiện ánh sáng giảm nhẹ — chẳng hạn ảnh được chụp vào buổi chiều tối hoặc trong môi trường thiếu sáng — mô hình bắt đầu gặp khó khăn. YOLO11 có thể bỏ sót người trong ảnh, nhận nhầm đối tượng, hoặc thậm chí không tạo ra bất kỳ bounding box nào.

Điều quan trọng ở đây không phải là mô hình "trở nên tệ đi", mà là **dữ liệu đầu vào đã thay đổi**. Ánh sáng giảm làm brightness thấp hơn, noise tăng lên, độ sắc nét của ảnh giảm, và đôi khi hình còn bị che khuất (occluded). Tất cả những yếu tố này đều khiến mô hình không còn làm việc trong điều kiện giống như lúc được huấn luyện.

Một hệ thống ML tốt phải theo dõi được những thay đổi này. Nó phải có khả năng cảnh báo khi brightness trung bình của ảnh đầu vào giảm bất thường, khi noise tăng lên, khi tỷ lệ ảnh mờ vượt quá ngưỡng, hoặc khi nhiều ảnh bị che khuất khiến mô hình khó nhận diện. Những tín hiệu đó cho phép nhóm vận hành hiểu rằng vấn đề không nằm ở mô hình, mà nằm ở dữ liệu.



Hình 1: Ví dụ theo dõi dữ liệu sau khi mô hình deploy

Nếu không có tracking, chúng ta hoàn toàn không biết vì sao hiệu suất mô hình suy giảm.

Do đó, trong MLOps, sau khi triển khai mô hình (deployment), bước tiếp theo *luôn luôn* là thiết lập một hệ thống giám sát bài bản.

2 Thực hành Monitoring cốt lõi trong MLOps

2.1 Tầm quan trọng của Monitoring

Phát hiện sớm vấn đề

Monitoring cho phép phát hiện sớm:

- Suy giảm hiệu năng mô hình (accuracy, F1, AUC, ...).
- Lỗi pipeline dữ liệu (ETL hỏng, dữ liệu thiếu cột, kiểu dữ liệu sai).
- Lỗi hạ tầng (hết RAM, đầy ổ đĩa, timeout, lỗi mạng).

Càng phát hiện sớm, ta càng giảm được rủi ro ảnh hưởng đến người dùng và chi phí sửa lỗi.

Theo dõi sử dụng tài nguyên

Các chỉ số như CPU, GPU, RAM, disk I/O cho biết hệ thống có đang vận hành *hiệu quả* hay không. Dựa vào đó, ta có thể:

- Quyết định *scale up* (máy mạnh hơn) hay *scale out* (thêm instance).
- Tối ưu code, giảm lãng phí tài nguyên.

Phát hiện Data Drift

Khi phân phối dữ liệu đầu vào thay đổi, mô hình dễ bị giảm chính xác. Monitoring giúp:

- So sánh phân phối dữ liệu mới với dữ liệu huấn luyện.
- Đưa ra cảnh báo khi drift vượt ngưỡng.
- Quyết định thời điểm cần retrain mô hình.

Giữ được tính giải thích (Explainability)

Trong nhiều ngành như tài chính, y tế, bảo hiểm, mô hình phải "giải thích được" (tại sao từ chối khoản vay, tại sao gợi ý phác đồ điều trị, …). Monitoring giúp đảm bảo:

- Mô hình không trôi dần sang hành vi khó giải thích.
- Ta có log và metrics để truy vết quyết định của mô hình theo thời gian.

2.2 Các nguyên tắc của một hệ Monitoring tốt

Functional Monitoring

Tập trung vào **hiệu năng chức năng** của mô hình:

- Accuracy, Precision, Recall, F1, ROC–AUC (cho classification).
- MAE, MSE, RMSE, R^2 (cho regression).
- Top–K accuracy, BLEU, ROUGE, …(cho NLP, recommender, v.v.).

Những chỉ số này cần được theo dõi theo thời gian, theo phiên bản mô hình, hoặc theo phân khúc người dùng.

Operational Monitoring

Tập trung vào **sức khoẻ hệ thống**:

- Throughput (số request/giây).
- Latency (95th, 99th percentile).
- Error rate (tỉ lệ 4xx, 5xx).
- Uptime, số instance đang hoạt động.

Functional tốt nhưng Operational tệ ⇒ người dùng vẫn có trải nghiệm xấu.

Scalable Integration

Khi hệ thống ML mở rộng (nhiều model, nhiều service, nhiều vùng địa lý), hệ thống monitoring cũng phải scale theo:

- Hỗ trợ nhiều data source.
- Gánh được lượng metrics lớn.
- Dễ bổ sung thêm dashboard, alert, service mới.

Tracking & Lưu trữ chỉ số

Cần lưu trữ metrics một cách có cấu trúc (thường là time-series database):

- Dễ so sánh theo thời gian (trước/sau deploy, theo mùa vụ).
- Dễ phân tích sâu (ví dụ: join với log, với dữ liệu business).

Alerting

Một hệ monitoring tốt không chỉ *hiển thị*, mà còn phải *báo động*:

- Gửi email, Slack, Teams, PagerDuty khi metric vượt ngưỡng.
- Cho phép định nghĩa ngưỡng theo phần trăm, theo quantile, theo *rate of change*.

3 Các nhóm Metrics quan trọng cần theo dõi

3.1 Server Metrics

Đây là lớp thấp nhất, đảm bảo hạ tầng chạy ổn.

- **CPU/GPU Utilization**: nếu luôn ở 95–100%, có thể cần scale.
- **Memory Usage**: nếu gần chạm trần, dễ bị OOM (out of memory).
- **Disk I/O & Storage**: đọc/ghi chậm, ổ đầy, log không ghi được.
- **Availability (Uptime)**: tỉ lệ thời gian service sẵn sàng.
- **Latency**: thời gian xử lý mỗi request, đặc biệt là latency của bước inference mô hình.
- **Throughput**: số request/giây, dùng để kiểm tra khả năng chịu tải.

3.2 Input Data Metrics

Đảm bảo mô hình nhận đúng "chất liệu" đầu vào.

- **Phiên bản mô hình (Model Version)**: cần log rõ request đang dùng version nào để so sánh hiệu năng.
- **Data Drift**: so sánh phân phối của dữ liệu online với dữ liệu train.
- **Outlier Detection**: phát hiện điểm dị biệt (quá lớn, quá nhỏ, giá trị chưa từng thấy).
- **Tỉ lệ missing**: số lượng giá trị null/NaN, tỉ lệ field thiếu.

Ví dụ với ảnh:

- Brightness trung bình.
- Độ tương phản.
- Tỉ lệ ảnh bị blur.

3.3 ML Model Metrics

Phần này đo **chất lượng dự đoán**.

Classification

- Accuracy, Precision, Recall, F1.
- ROC–AUC, PR–AUC.
- Confusion matrix theo thời gian.

Regression

- MAE, MSE, RMSE, R^2.
- So sánh phân phối y_{true} và y_{pred}.

Prediction Monitoring & Prediction Drift

Ngoài input drift, ta còn phải theo dõi:

- Phân phối của dự đoán (prediction distribution).
- Tỉ lệ lớp (class proportion) trong dự đoán.
- Tần suất mô hình trả về "null" hoặc "không chắc chắn".

Khi phân phối dự đoán thay đổi mạnh theo thời gian ⇒ **prediction drift**.

4 Drift trong Machine Learning

Drift là thay đổi theo thời gian trong dữ liệu, dự đoán hoặc mối quan hệ giữa đầu vào và đầu ra. Đây là nguyên nhân lớn khiến mô hình *suy giảm* sau khi deploy.

4.1 Data Drift

Data Drift xảy ra khi phân phối thống kê của dữ liệu đầu vào thay đổi so với lúc huấn luyện. Ví dụ:

- Trang phục, ánh sáng, bối cảnh trong ảnh camera an ninh thay đổi theo mùa.
- Hành vi khách hàng online thay đổi trong dịp lễ, Tết.
- Ngôn ngữ, từ lóng mới xuất hiện trong ứng dụng chat.

Cách xử lý:

- Theo dõi phân phối (histogram, quantile, mean, variance, …).
- Dùng kiểm định thống kê (KS-test, chi–square, …).
- Khi drift vượt ngưỡng ⇒ retrain mô hình với dữ liệu mới.

4.2 Prediction Drift

Prediction Drift là thay đổi trong *dự đoán* của mô hình theo thời gian, dù input không thay đổi quá nhiều.

Ví dụ:

- Tỉ lệ mô hình chấm "rủi ro cao" cho khách vay đột ngột tăng.
- Tỉ lệ model đề xuất một nhóm sản phẩm nào đó tăng bất thường.

Ta có thể phát hiện bằng cách:

- Theo dõi Accuracy/Precision/Recall theo thời gian.
- Theo dõi phân phối lớp dự đoán (class proportion).

4.3 Concept Drift

Concept Drift xảy ra khi mối quan hệ $X \to y$ thay đổi. Khác với data drift (thay đổi phân phối X), concept drift là thay đổi *bản chất khái niệm*.

Ví dụ:

- Mô hình dự báo giá nhà được train trước Covid; sau Covid, hành vi và giá nhà thay đổi.
- Mô hình chẩn đoán bệnh dựa trên guideline cũ; guideline y khoa mới được cập nhật.

Xử lý concept drift thường khó hơn, cần:

- Retrain định kỳ.
- Sử dụng các mô hình thích nghi online (online learning, adaptive algorithms).
- Thiết lập cơ chế tự động kích hoạt retrain khi nghi ngờ drift.

5 Prometheus & Grafana: Bộ đôi kinh điển trong Monitoring

Phần này kết hợp nội dung từ slide AI Vietnam và bài viết "Prometheus & Grafana".

5.1 Prometheus là gì?

Prometheus là bộ toolkit **monitoring & alerting** mã nguồn mở, chuyên thu thập và lưu trữ metrics dạng **time series**. Đặc điểm:

- Mô hình dữ liệu đa chiều (metric name + cặp nhãn key/value).
- Lưu trữ cục bộ, không phụ thuộc hệ thống phân tán.
- Thu thập metrics theo cơ chế **pull**: Prometheus đi "scrape" endpoint `/metrics`.
- Có ngôn ngữ truy vấn mạnh mẽ **PromQL**.

5.2 Grafana là gì?

Grafana là nền tảng **quan sát & trực quan hóa** mã nguồn mở, chuyên dùng để:

- Tạo dashboard đẹp.
- Hiển thị biểu đồ, bảng, gauge, heatmap, …
- Đặt cảnh báo (alert) dựa trên metrics.
- Kết nối nhiều nguồn dữ liệu: Prometheus, Loki, Elastic, SQL, …

Prometheus thu thập & lưu metrics; Grafana hiển thị & trực quan hóa. Đây là lý do hai công cụ thường đi cùng nhau.

5.3 Vì sao dùng Prometheus & Grafana cho MLOps?

- **Real–time monitoring**: theo dõi latency, throughput, error rate, drift score, …theo thời gian thực.
- **Insight sâu**: có thể đào sâu từng service, từng mô hình, từng phiên bản model.
- **Scalability**: thiết kế cho hệ thống lớn, nhiều microservice.
- **Debug nhanh**: dashboard trực quan giúp khoanh vùng lỗi nhanh hơn.

6 Hướng dẫn sử dụng Prometheus & Grafana cho MLOps

Phần này dựa trên mục "Using Prometheus & Grafana" trong bài gốc, được chuyển sang LaTeX và Việt hoá.

6.1 Thiết lập Prometheus để giám sát Python/ML

Instrument Python Code

Ta cần "gắn cảm biến" (instrumentation) vào ứng dụng Python để Prometheus có thể scrape metrics.

Ví dụ đơn giản:

```
from prometheus_client import start_http_server, Summary
import random
import time

# Metric theo dõi thời gian xử lý request
REQUEST_TIME = Summary(
    'request_processing_seconds',
    'Time spent processing request'
)

# Trang trí hàm bằng metric
@REQUEST_TIME.time()
def process_request(t):
    """Hàm giả lập xử lý mất thời gian."""
    time.sleep(t)
```

```
if __name__ == '__main__':
    # Mở HTTP server expose metrics tại port 8000
    start_http_server(8000)

    # Sinh request giả lập
    while True:
        process_request(random.random())
```

Prometheus sẽ định kỳ gọi tới `http://<PYTHON_SERVER_IP>:8000/` để đọc metrics.

Cấu hình Prometheus

Tạo file `prometheus.yml`:

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'python'
    static_configs:
      - targets: ['<PYTHON_SERVER_IP>:8000']
```

Thay `<PYTHON_SERVER_IP>` bằng IP hoặc hostname thật.

6.2 Thiết lập Grafana cho dashboard & alert

Chạy Grafana

Grafana có thể chạy bằng Docker:

```
docker run -d -p 3000:3000 grafana/grafana
```

Truy cập: `http://<GRAFANA_HOST>:3000`

Thêm Prometheus làm Data Source

Trong UI Grafana:

1. Vào **Configuration** → **Data sources**.
2. Thêm Prometheus.
3. Điền URL: `http://<PROMETHEUS_HOST>:9090`.

Tạo Dashboard

Bạn có thể tạo các panel như:

- Biểu đồ time-series cho `request_processing_seconds`.
- Gauge cho error rate.
- Bảng (table) hiển thị các instance và tình trạng.

Thiết lập Alert

Grafana cho phép đặt rule:

- Nếu latency p95 > 1s trong 5 phút ⇒ gửi cảnh báo.
- Nếu error rate > 5% trong 10 phút ⇒ ping on-call.

Alert có thể gửi qua email, Slack, Teams hoặc tích hợp với các hệ thống quản lý sự cố.

6.3 Docker hoá Prometheus & Grafana

Prometheus

Chạy Prometheus bằng Docker và mount file cấu hình:

```
docker run -d -p 9090:9090 \
  -v /path/to/prometheus.yml:/etc/prometheus/prometheus.yml \
  prom/prometheus
```

Grafana

Như ở trên:

```
docker run -d -p 3000:3000 grafana/grafana
```

Khi kết hợp hai container này (và ứng dụng ML của bạn), bạn đã có một hệ thống monitoring cơ bản cho MLOps:

- Python/ML app expose metrics.
- Prometheus scrape & lưu metrics.
- Grafana hiển thị & alert.

7 Kết luận

Trong hệ sinh thái MLOps hiện đại, **monitoring** không còn là "tuỳ chọn", mà là **yêu cầu bắt buộc** nếu ta muốn hệ thống ML vận hành ổn định, an toàn, và đáng tin cậy trong thời gian dài.

- Các thực hành monitoring tốt giúp phát hiện sớm lỗi, tối ưu chi phí và giữ vững chất lượng mô hình.
- Việc theo dõi đúng nhóm metrics (server, input, model, drift) là chìa khóa để hiểu hệ thống *thật sự* vận hành ra sao.
- Prometheus & Grafana là bộ đôi lâu đời, mạnh mẽ, linh hoạt cho bài toán này.

Khi đã quen với pipeline: *Instrument* → *Collect* → *Visualize* → *Alert*, bạn có thể mở rộng thêm:

- Logging với Loki.
- Tracing với Jaeger, Tempo, v.v.
- Tự động retrain khi drift vượt ngưỡng.

Đó chính là con đường từ một mô hình ML đơn lẻ đến một hệ thống **ML production** thực sự trưởng thành.