Development of Machine-learning based App for Anomaly Detection in CMSWEB

Nasir Hussain[1], Amna Muzaffar[2], Muhammad Imran[3], Andreas Pfeiffer[1], Aroosha Pervaiz[1]

and Valentin Kuznetsov[4] for CMS Collaboration

[1]CERN, Geneva, Switzerland
[2]Quaid-i-Azam University, Islamabad Pakistan
[3]National Centre for Physics, Islamabad Pakistan
[4]Cornell University, New York, USA

Email: muhammad.imran@cern.ch

Agenda

- **Overview & Motivation**
- **Data Collection**
  - Data Processing and Storage
  - Datasets Overview
- **Machine learning Models**
  - Models Overview
  - Models Implementation
  - Models Evaluation Metrics
- **Hyperparameter Tuning**
  - Overview
  - Comparison
- **Live Anomaly Detection**
- **Application Development**
  - Training
  - Monitoring
- **Application Deployment**
- **Alert Mechanism**
- **Conclusion & Future Work**

Overview & Motivation

Introduction

- The CMSWEB infrastructure hosts critical web services like DBS, DAS, CRAB, WMCore, DQM and more.
- Built on Kubernetes (k8s), powering over two dozen distinct web services vital for CMS operations.
- Any irregularities or performance degradation can significantly impact CMS services and operations.

Key Functionality

- **Anomaly Detection & Alerting:**
  - Once an anomaly is detected, the system generates real-time alerts and routes them to relevant service developers, ensuring rapid response and issue resolution.
- **Impact:**
  - Proactively fortifies the reliability and security of the CMSWEB cluster, reducing downtime and improving system stability.

Objective

- R&D for machine/deep learning algorithms for anomaly detection
- Develop a machine-learning base application to continuously monitor CMSWEB services, detecting and addressing anomalies that indicate performance issues or security threats
- Leverage machine/deep learning techniques to monitor key service parameters, identifying deviations from expected behavior.

Data Collection

Introduction

- **Objective**:

Extract critical metrics from services hosted on CMSWEB for anomaly detection.

- **Source**:

Data is collected from CMS Monit Infrastructure using PromQL

- **Target Services**:

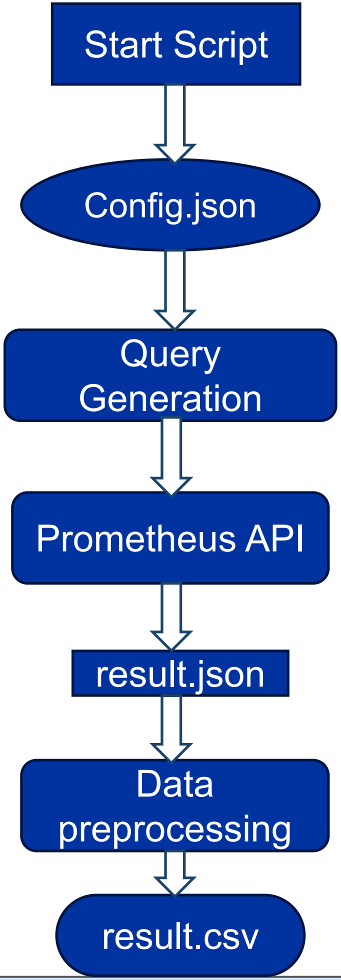
Monitored services include DBS, DAS, CRAB, WMarchive, and WMCore etc.

- **Metrics**:
  - CPU
  - Memory

Process

- Prometheus queries are dynamically generated for each service, namespace, and container.
- Time intervals:
  - Initial run uses a 30-day history.
  - Continuous training runs every 2 hours.

Data Processing and Storage



Data Processing:

- Prometheus data is processed to filter out irrelevant results.
- Data is cleaned and aggregated by timestamp, taking the mean value for each interval.

Memory Management for limited resources:

- Memory usage is logged at multiple stages.
- Garbage collection is explicitly called to manage large datasets.

Data Storage:

- Processed data is saved to CSV files for later use in machine learning model training.
- Results are continuously appended, ensuring up-to-date data for real-time anomaly detection.

Datasets Overview

Namespace & Container

Namespace	Containers
crab	crabserver
das	das-server
dbs	dbs2go-global-r
dmwm	reqmgr2
dqm	autodqm
wma	wmarchive
ruciocm	monitor
dmwm	workqueue
tzero	t0wmadatasvc

Metrics

Name
eagle_pod_container_resource_usage_memory_bytes
eagle_pod_container_resource_usage_cpu_cores

Models Overview

Key Points

- All models are designed using Autoencoder architecture to detect anomalies by learning data reconstruction patterns.
- Autoencoders are used to identify deviations by comparing input data to its reconstruction.

Model Implemented

1. **CNN Autoencoder**: Extracts spatial features from input data using convolutional layers.
2. **LSTM Autoencoder**: Captures long-term dependencies and temporal patterns in sequential data.
3. **GRU Autoencoder**: A simplified version of LSTM with fewer parameters, designed for faster training while preserving temporal patterns.
4. **Fully-Connected Autoencoder**: Uses dense layers for a basic autoencoder architecture to model general patterns.
5. **Hybrid CNN-LSTM Autoencoder**: Combines convolutional layers for spatial feature extraction with LSTM layers for temporal analysis, offering enhanced performance for time-series data.
6. **Hybrid CNN-GRU Autoencoder**: Merges CNN for feature extraction and GRU for sequence learning, offering a balance between performance and efficiency.

Models Implementation

Data Handling

Data Loading:

- CSV files containing metrics are loaded

Normalization:

- Data is normalized to a range suitable for model input

Sequence Generation:

- Sequences are created from the data for training and testing.

Evaluation Metrics

Performance Metrics:

- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)
- Mean Absolute Squared Error (MASE)
- R^2 Score

Visualization:

- Plots of reconstruction errors to assess model performance and detect anomalies.

Anomaly Detection Process

Isolation Forest:

- Applied to detect and filter anomalies in the dataset before training.
- Masks anomalies for subsequent model training.

Model Training:

- **Callbacks**: Implements EarlyStopping and ModelCheckpoint for better training performance.

Continuous Training and Monitoring

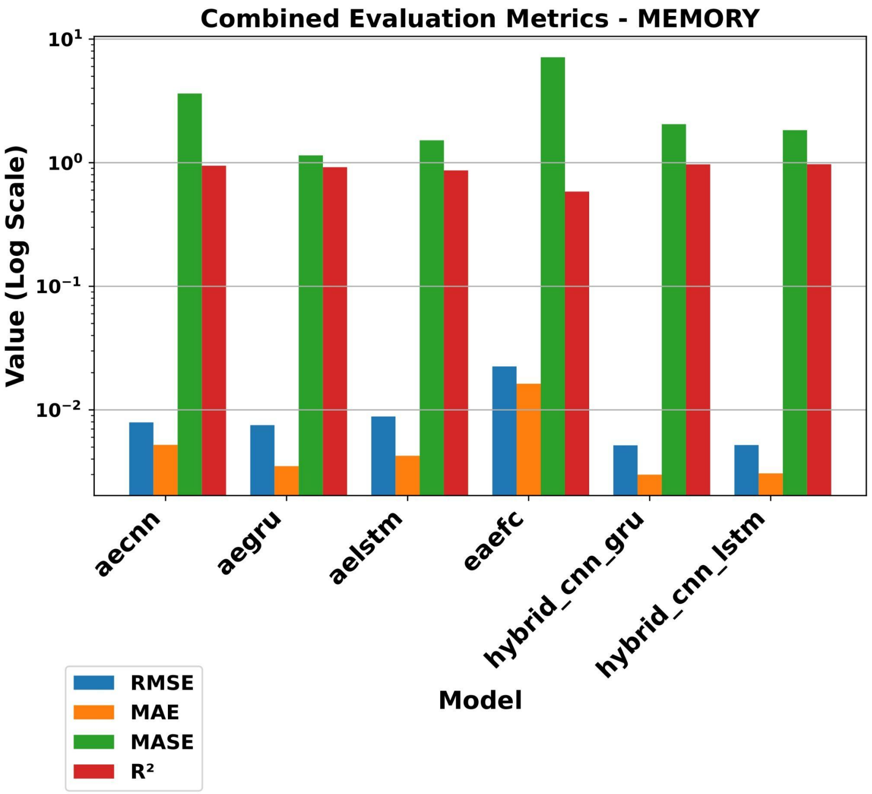
Implementation of Continuous Training:

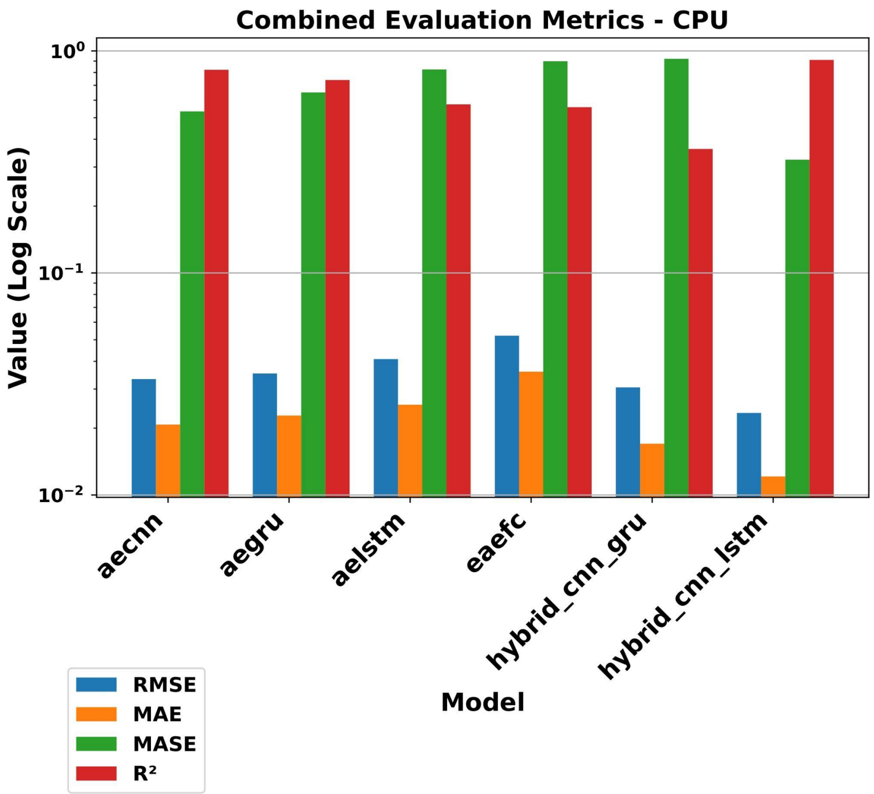
- Allows the model to adapt to new data patterns without manual intervention.

Real-Time Monitoring:

- Regular checks for new data and updates to the training process.

Model Evaluation – Training Data - Error Metrics





Model Evaluation Metrics

Thresholds

1. **Mean + k * Standard Deviation**

Formula: threshold_mean_std=avg_reconstruction_error_train+ k×std(reconstruction_error_train)

Description: Establishes a threshold based on the average reconstruction error and its variability, scaled by a factor k.

2. **Median Absolute Deviation (MAD)**

Formula:

threshold_mad=median(reconstruction_error_train)+k ×mad

Description: Focuses on the robustness of the median and its variability, providing a threshold based on the dispersion of reconstruction errors.

1. **95th Percentile**

Formula:

threshold_95th_percentile=percentile(reconstruction_error_train,95)

Description: Represents the value below which 95% of the reconstruction errors fall, marking a high anomaly threshold.

2. **99th Percentile**

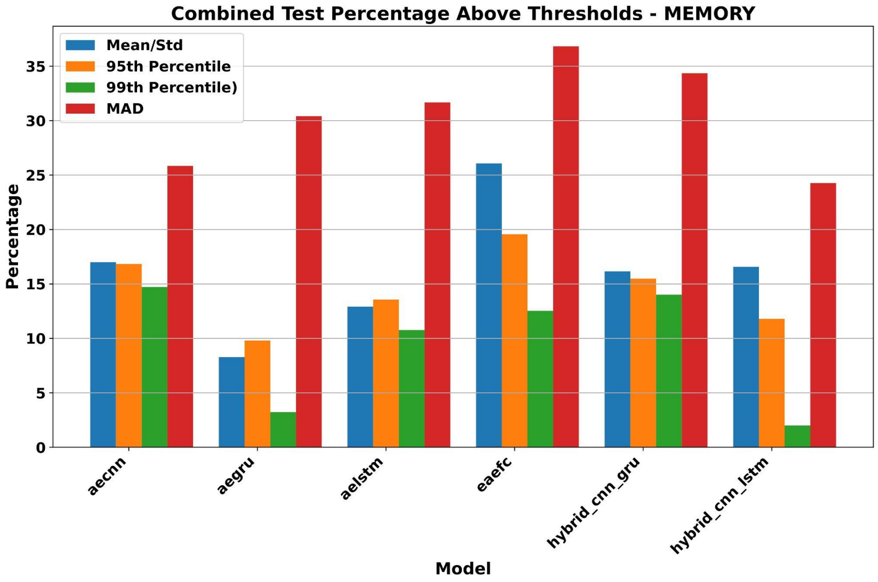
Formula:

threshold_99th_percentile=percentile(reconstruction_error_train,99)

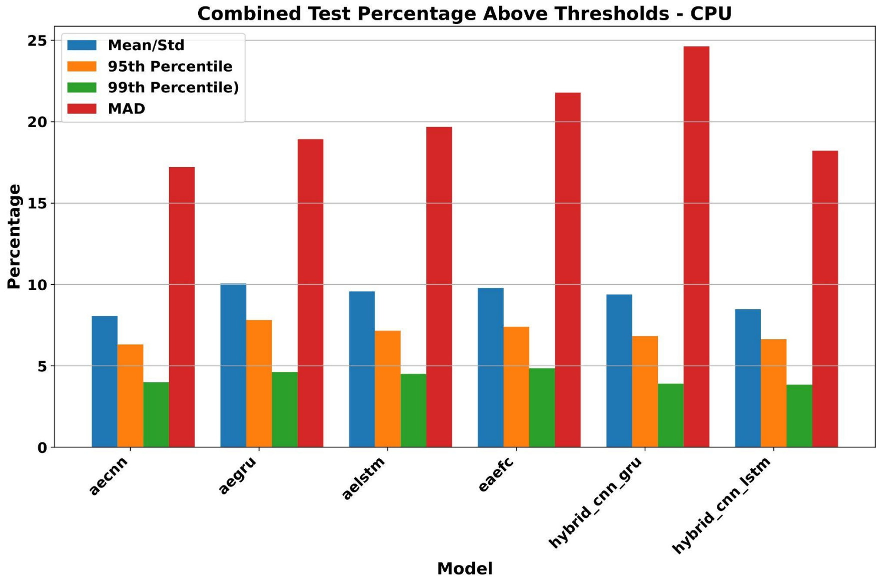
Description: Indicates a more stringent threshold for anomaly detection, capturing only the top 1% of reconstruction errors.

Model Evaluation – Test - %age Above Thresholds

Memory Datasets

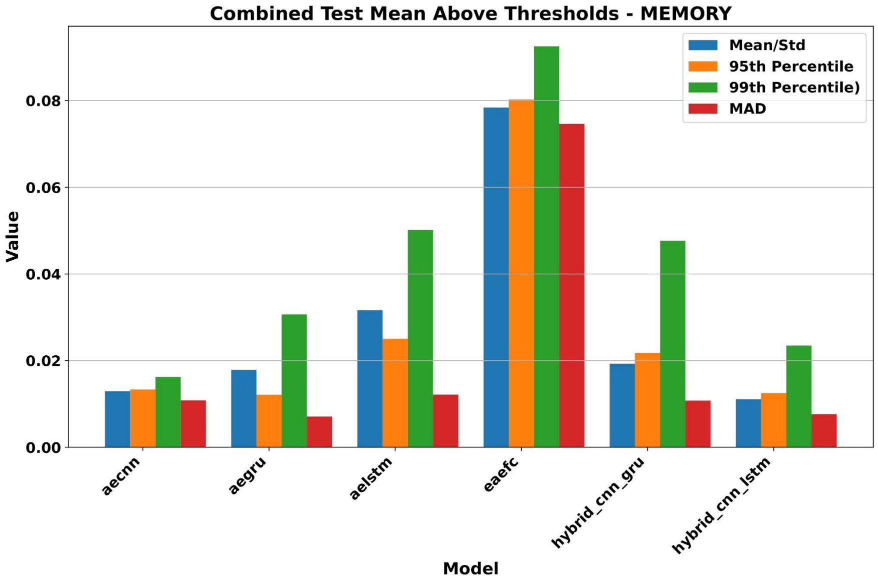


CPU Datasets

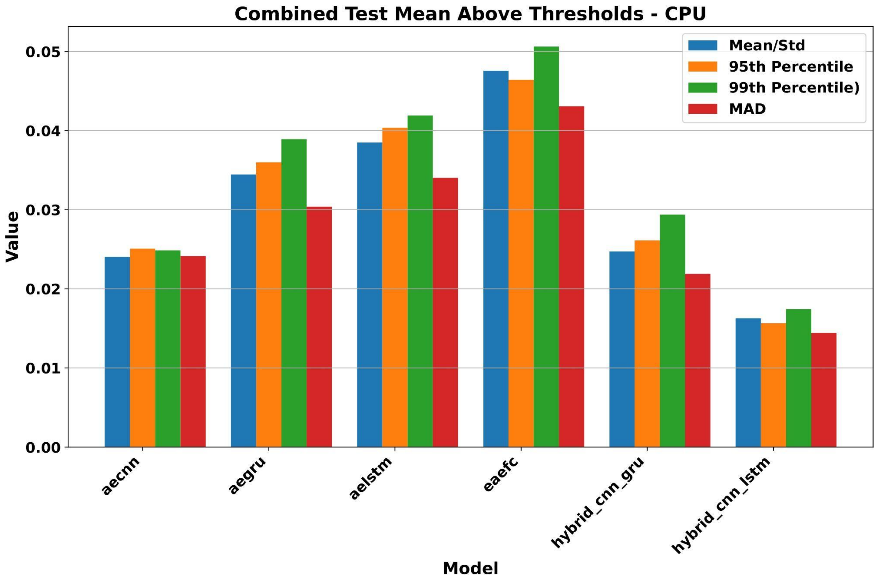


Model Evaluation – Test - Mean Above Thresholds

Memory Datasets

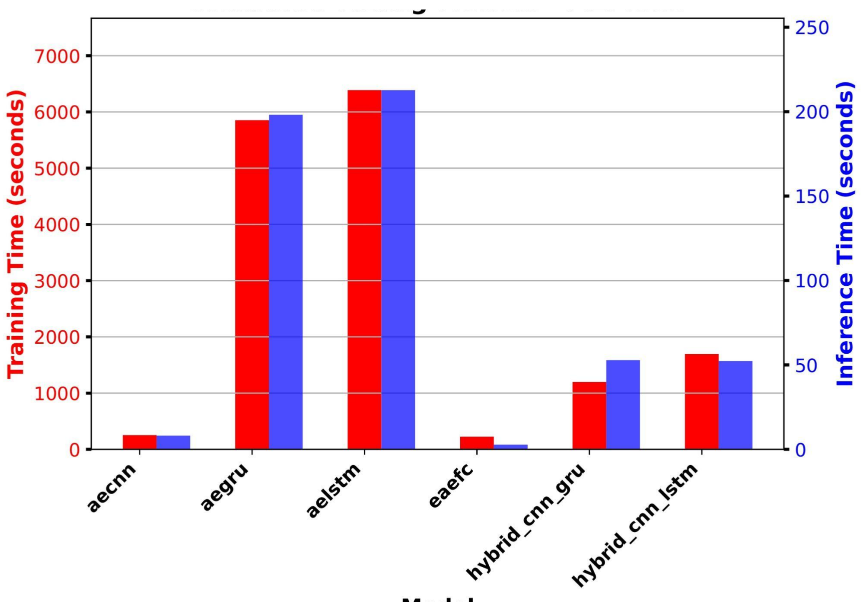


CPU Datasets

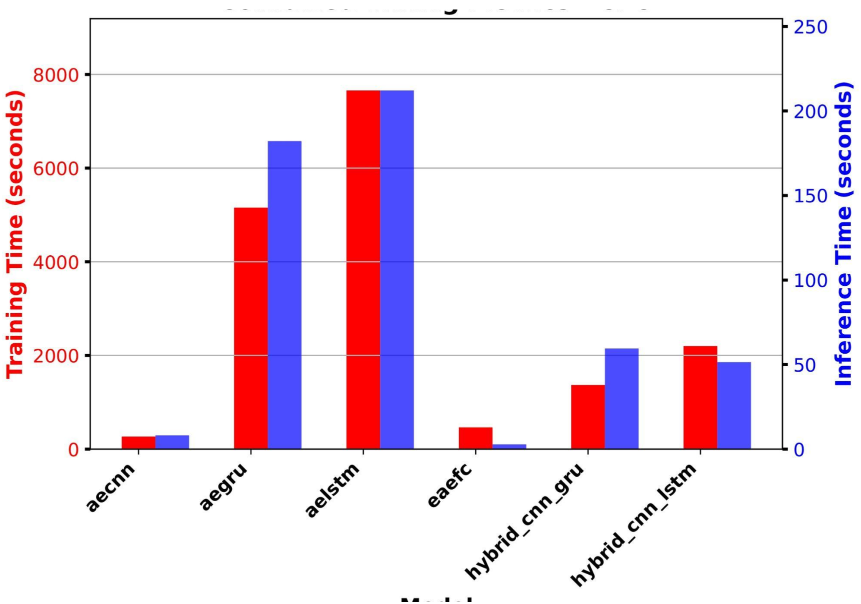


Model Evaluation – Training & Inference Time

Memory Datasets



CPU Datasets



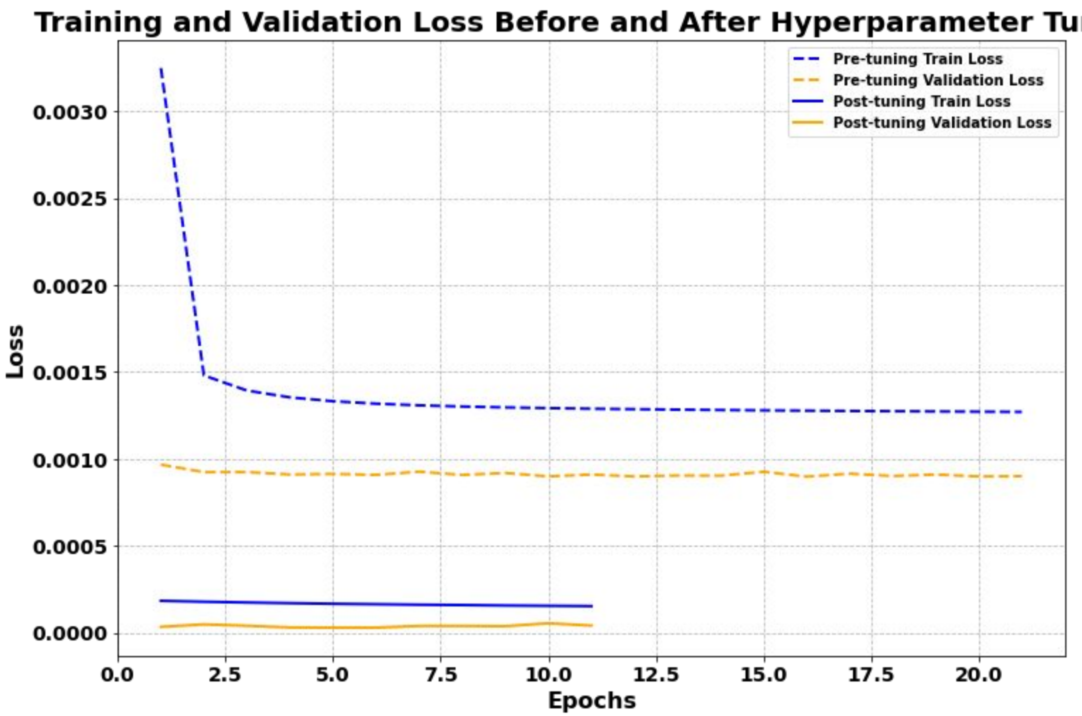
Hyperparameter Tuning Overview

Search Strategy:
Implement a search strategy, such as:
- **Random Search** for efficient exploration of the hyperparameter space.

Callbacks:
Use callbacks for model training:
- **EarlyStopping**: Monitors validation loss to prevent overfitting.
- **ModelCheckpoint**: Saves the best model based on validation loss.

Tuning Execution:
- Initialize a tuner (e.g., Keras Tuner) with:
  - Maximum trials for hyperparameter combinations.
  - Number of executions per trial for robustness.

Hyperparameter Tuning Comparison



Live Anomaly Detection

Objective

- Dynamically adjust thresholds over time to improve anomaly detection accuracy as data evolves.

•**Adaptive Thresholds** continuously update based on:

1. **Historical Thresholds**: Derived from previous data.
2. **Recent Data**: Latest training results and reconstruction errors.
3. **Time Decay**: Older thresholds are given less importance over time.

$$T_{\text{adaptive}} = \left(1 - e^{-\lambda \cdot t_{\text{diff}}}\right) \cdot \left(w \cdot T_{\text{prev}} + (1 - w) \cdot T_{\text{recent}}\right) + e^{-\lambda \cdot t_{\text{diff}}} \cdot T_{\text{recent}}$$

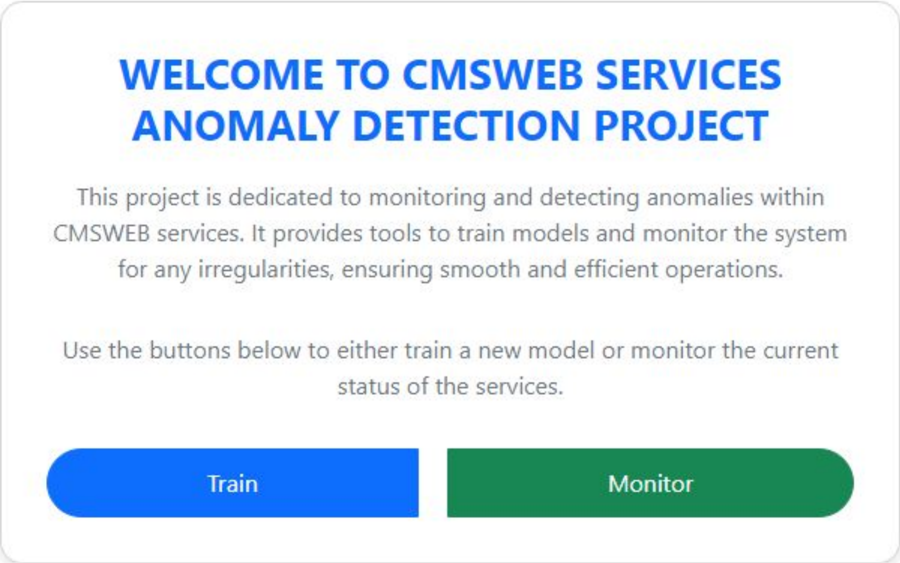
$$w = \frac{\sigma_{\text{prev}}}{\sigma_{\text{prev}} + \sigma_{\text{recent}}}$$

Outcome:

Real-time adaptation of thresholds improves detection of evolving anomalies, making the system more responsive to changes in data.

Application Development

Welcome Page: https://cmsweb-anomaly-detection.app.cern.ch



Application: Training

https://cmsweb-anomaly-detection.app.cern.ch/train/

- It is used to train various applications by selecting various filters

CMSWEB SERVICES ANOMALY DETECTION TRAINING DASHBOARD

Monitor

Select Parameters

Env:

k8s-prod

Namespace

crab

Application

Metric:

eagle_pod_container_resource_usage_memory_bytes

Models:

HybridCnnLstm

Train

Training Applications

Env	Model	Namespace	Container	Metric	Start Time	Status
k8s-prod	HybridCnnLstm	crab	crabserver	eagle_pod_container_resource_usage_memory_bytes	2024-09-26 15:24:07	Running

Application: Monitoring

https://cmsweb-anomaly-detection.app.cern.ch/monitor/

- It is used to monitor various applications by selecting various filters

CMSWEB SERVICES ANOMALY MONITORING DASHBOARD

Train

Select Parameters

Environment:

k8s-prod

Model

Select Model

Metric

Select Metric

Namespace

Select Namespace

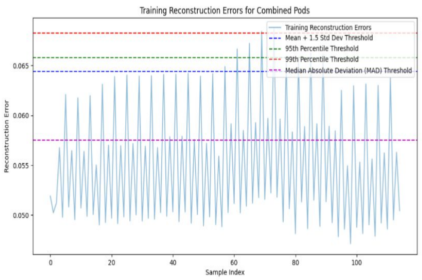
Application

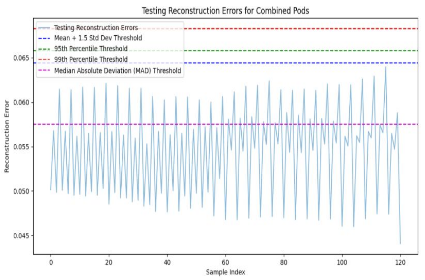
Select Application

Time

Select Time

Monitor



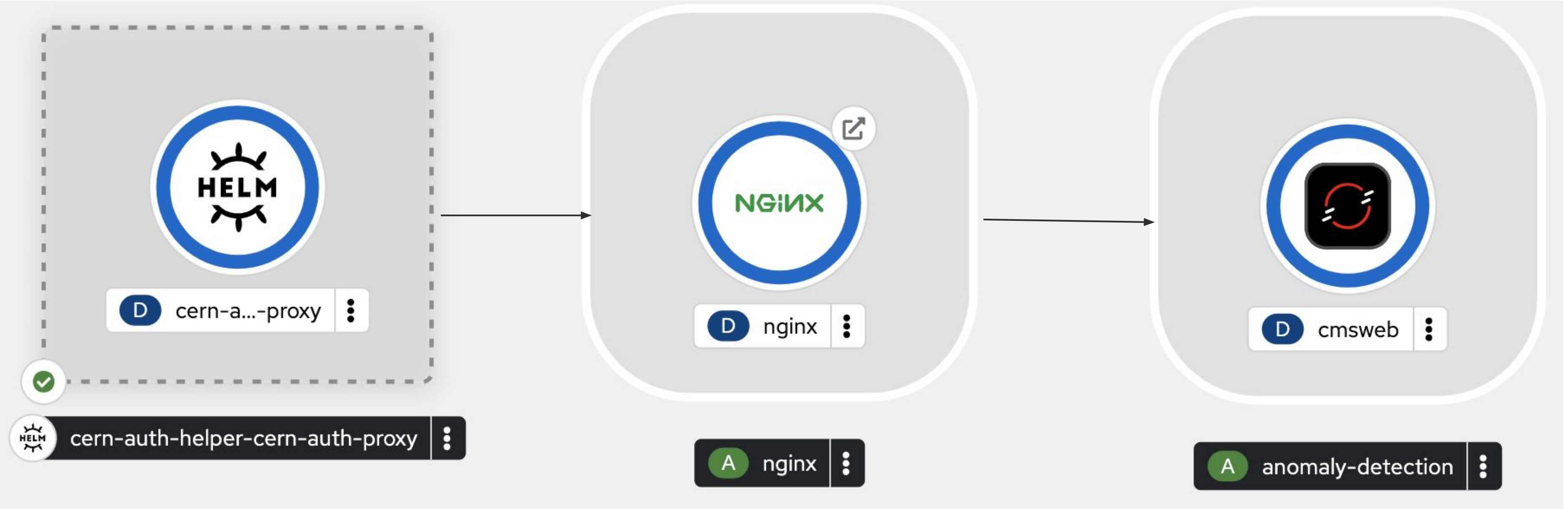


Test Percentage Above Threshold

Deployment

CERN Platform as a service (Paas)

- Deployment is done on CERN Openshift Infrastructure.
- CERN SSO to authentication
- Nginx Proxy to provide role based access based on egroups



Alert Mechanism

- Alerts are generated using amtool in CMS Monit Infrastructures based on rules
- Email is also sent to receiver set for particular tag

− default | Not grouped | 1 alert

2024-10-02T10:26:28.804Z − Info Silence Link

date: 2024-10-02 10:21:28

hostname: https://cmsweb-anomaly-detection.app.cern.ch/monitor/?env=k8s-prod&model=HybridCnnLstm&metric=eagle-pod-container-resource-usage-memory-bytes&namespace=dmwm&container=reqmgr2&time=2024-10-02T10:21:28.423719Z

message: Anomaly Detected:critical

alert="amtool" + env="k8s-prod" + metric="eagle-pod-container-resource-usage-memory-bytes" + service="reqmgr2" +
severity="critical" + tag="dmwm" +

Conclusion & Future Work

Conclusion

- Performed R&D for anomaly detection for web services deployed in CMSWEB services.
- Data extraction from CMS Monit Infrastructure
- Implemented various machine learning algorithms and monitored performance against various applications.
- Performed hyper parameter tuning
- Development of App to train app and detect anomalies and generate alerts
- Deployment on Openshift platform.

Future Work

- Improvement in monitoring dashboard
- Including more anomalies insight like count of anomalous points at certain timestamps for more effective decision making.
- Allow users to directly interact with the model such as allowing them to set certain parameters for model training
- Create a summarized dashboard about applications that are in training and those that need to be trained and their status.
- Allow the user to stop the currently running model.
- Implement an expert feedback loop that will be used to ignore the anomalous point in the next round of model training.

Thank You.
Q/A

Backup slides

Model Evaluation Metrics

Evaluation Metrics

1. Root Mean Squared Error (RMSE)

- Measures the square root of the average squared differences between predicted and actual values.
- **Interpretation**: Lower RMSE indicates better fit, as it reflects how close predictions are to actual data points.

2. Mean Absolute Error (MAE)

- Represents the average absolute differences between predicted and actual values.
- **Interpretation**: Lower MAE signifies better predictive accuracy, indicating smaller errors between predicted and actual values.

3. Mean Absolute Scaled Error (MASE)

- Scales the MAE relative to the MAE of a naïve baseline model.
- **Interpretation**: MASE < 1 indicates that the model performs better than the baseline; MASE > 1 indicates worse performance.

4. R-squared (R^2)

- The proportion of the variance in the dependent variable that is predictable from the independent variables.
- **Interpretation**: R^2 ranges from 0 to 1, where values closer to 1 indicate a better fit of the model to the data.

Hyperparameter Tuning Overview

Objective

- Optimize model performance by selecting the best hyperparameters through systematic search.

Model Definition:
Use a flexible model architecture (e.g., Hybrid CNN-LSTM) that allows tuning of various hyperparameters such as:
- Number of filters in CNN layers.
- Kernel size for convolution operations.
- Number of units in LSTM layers.
- Dropout rates for regularization.
- Choice of optimizer (e.g., Adam, RMSprop).

Hyperparameter Space:
Define a range for each hyperparameter:
- Filters: [64, 128, 192, 256]
- Kernel Size: [2, 3, 4, 5]
- LSTM Units: [64, 128, 192, 256]
- Dropout Rate: [0.1, 0.2, 0.3, 0.4, 0.5]