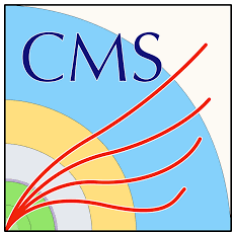
Conseil Européen pour la Recherche Nucléaire - CERN



Compact Muon Solenoid - CMS

Development of machine-learning based app for anomaly detection in CMSWEB

November 25, 2024

Internship Report

Nasir Hussain (CERN Summer Student - 24)

Supervised by:
Muhammad Imran
Andreas Pfeiffer

Abstract

This project that we have developed revolves around the idea of an advanced anomaly detection system for the CMSWEB services, which is a vital part of CERN's infrastructure that houses more than two dozen web services like DBS, DAS, CRAB, and WMCore. We aimed to improve these services' performance and reliability by detecting anomalies in real-time with the help of machine learning (ML) and deep learning (DL) models. Several autoencoder-based models like Hybrid CNN-LSTM, Hybrid CNN-GRU, GRU, LSTM, CNN, and a fully connected autencoder were trained by processing the data gathered from the CMS Monit infrastructure.

There are multiple thresholds of anomaly detection incorporated in this system based on statistical methods, like mean + 1.5 standard deviations, Median absolute deviation (MAD) and the 95th and 99th percentiles of errors in reconstruction. To detect the deviations in the testing data, the above mentioned thresholds were applied which provided a robust procedure for anomaly detection. The Hybrid CNN-LSTM model demonstrated as a reliable and suitable model for anomaly detection in CMSWEB services by consistently outperforming the other models in regards to root mean squared error (RMSE), Mean Absolute Error (MAE), and R-squared.

The platform used to deploy the system was OpenShift, which provided real-time insights, visualizations and alerts that were automated for anomalies related to services. We used hyper-parameter tuning and continuous learning approaches to ensure the adaption of the models to the changing services behaviors. This project holds significant potential for enhancing CERN's operational efficiency by reducing downtime and improving system stability. Future work will focus on improving the monitoring dashboard, expanding anomaly insights, integrating user interaction features for model training, and implementing an expert feedback loop to refine the detection process.

Keywords: Anomaly Detection, Machine Learning, Deep Learning, CNN-LSTM, CMSWEB, Autoencoder, Real-time Monitoring, Thresholds, OpenShift, AI-driven Monitoring, CERN

Contents

1 Introduction

1.1 Context and Motivation

Anomaly detection is a critical aspect of monitoring and maintaining the reliability of complex systems. The Compact Muon Solenoid (CMS) experiment, one of the major detectors at the Large Hadron Collider (LHC) at CERN, generates and processes vast amounts of data, necessitating robust monitoring solutions to ensure high performance and operational stability [17]. The CMSWEB cluster supports the CMS experiment's infrastructure by hosting over two dozen essential web services, including DBS, DAS, CRAB, and WMCore [17]. This cluster operates on Kubernetes, providing scalable and reliable services critical to CMS's daily operations [17]. Architecture of CMSWEB cluster is shown in Figure 1.

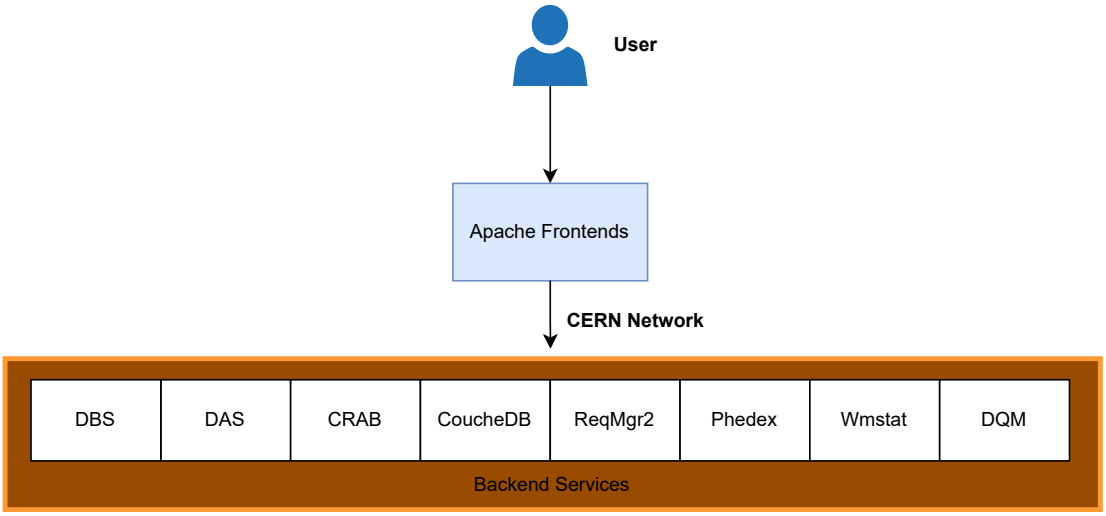


Figure 1: The CMSWEB cluster consists of two layers, i.e. the front-end and back-end services. Adapted from [17]

In the context of CMSWEB services [2] and the CMS experiment [10], anomaly detection is particularly significant due to the large-scale and critical nature of the infrastructure supporting the CMS experiment. These services are integral to the smooth operation of the experiment, and any disruptions or anomalies can lead to significant impacts on the overall performance and stability of the experiment. For ensuring the timely identification, resolution of problems, limiting potential downtime and improving the resilience of the CMSWEB infrastructure, automated anomaly detection solutions are necessary.

1.2 Challenges

There are several challenges associated with the handling of large-scale datasets and anomaly detection in complex systems. Vast amounts of times series data is generated by the CMSWEB cluster which are related to performance metrics like CPU and usage of memory. This data requires a robust system to be managed and analyzed with that has the capability to process and interpret complex patterns. The interplay of number of factors and dynamic nature of the services makes it quite complicated to identify anomalies in such environments. The traditional methods lack in regards to addressing

the above mentioned challenges, hinting a need for an advanced approach that is AI-driven and has the ability to handle huge volumes of data and detect large volumes of data and identify indistinct anomalies from the normal or routine behavior [23]. Methods that are traditional like simple thresholding, statistical process control (SPC), and rule-based approaches, fall short to fulfill the demands of systems that are highly complex with non-linear behavior and extensive datasets typically depend on rules that are predefined or assumed stationary behavior, which are not adaptable with the CMSWEB like dynamic systems. In contrast to the traditional methods, the AI-driven advanced techniques like machine learning and deep learning models, provide more robust solutions [23]. Subtle deviations from expected behaviors and large data volumes are easily handled by these models particularly autoencoders and hybrid architectures which are designed for it.

1.3 Objective

This research has a particular objective which is to design and develop an application that is AI-based for real-time detection of anomalies in the CMSWEB cluster. The aim of this proposed solution is to present an uninterrupted monitoring of critical services for anomaly detection and automated alerts generation for those detected anomalies. Various AI models for analyzing time series data and identifying potential issues will be incorporated in the application aiming to enhance the CMSWEB insfrastructure‘s stability and reliability.

1.4 Contributions

The study contributes significantly to the field of monitoring and detection of anomalies:

- **Exploring number of AI models:** Various AI models such as autoencoders, LSTM (Long short-term memory), CNN (Convolutional Neural Network), GRU (Gated Recurrent Unit), and hybrid models combining CNN with LSTM and GRU are comprehensively evaluated [11] [28].
- **Selection of the Best Hybrid Model:** Through comparative analysis based on multiple evaluation metrics, the Hybrid CNN-LSTM model was identified as the most effective approach for anomaly detection, while the Autoencoder Fully-Connected model was found to be the least effective.
- **Hyperparameter Tuning:** After selecting the best model we applied hyperparameter tuning using random search to enhance the model's capabilities.
- **Development of the Application:** The creation of a robust AI-driven application that provides real-time monitoring and alert generation. The application supports continuous data collection, model training, and anomaly detection, ensuring timely identification and notification of critical issues.

1.5 Paper Organization

The remainder of the paper is organized as follows:

- **Section 2: Related Work** – Reviews existing research in anomaly detection for large-scale infrastructures and discusses relevant AI approaches, comparing them with the methods used in this research.

- **Section 3: Dataset and Infrastructure** – Explains the data collection process using PromQL queries from CERN's CMS Monit infrastructure, the preprocessing steps, and provides an overview of the CMSWEB services.
- **Section 4: AI Algorithms Explored** – Describes the architectures and applications of the different AI models explored, including Autoencoder LSTM, CNN, GRU, fully-connected, and hybrid models.
- **Section 5: Model Selection and Hyperparameter Tuning** – Discusses the selection process for the best model, hyperparameter tuning for optimizing performance, and the evaluation metrics used.
- **Section 6: AI-based Application Design and Architecture** – Details the design of the AI-based application, its continuous data flow, real-time training, anomaly detection, web interface, and alert mechanisms.
- **Section 7: Results and Discussion** – Presents performance results, insights from the CMSWEB data, and showcases visualizations and insights generated by the application, along with the impact on CERN's operations.
- **Section 8: Conclusion** – Summarizes key findings, discusses the implications of the research on anomaly detection in high-performance infrastructures, and proposes future directions for improving and scaling the solution.

2 Related Work

Anomaly detection is crucial for ensuring the reliability and security of large-scale infrastructures, such as cloud-based services and microservice architectures. Traditional anomaly detection techniques often struggle in such complex environments due to the vast amount of data generated, the heterogeneity of data sources, and the lack of labeled datasets. To address these challenges, recent research has increasingly focused on AI-driven approaches for detecting anomalies in time-series data. This section reviews existing work in anomaly detection for large-scale infrastructures, discusses the application of AI algorithms in anomaly detection for CMSWEB services, and compares these approaches with the one presented in this work.

2.1 Anomaly Detection in Large-Scale Infrastructures

Anomaly detection in large-scale infrastructures presents significant challenges, particularly due to the lack of labeled data, which limits the use of supervised learning techniques. Resultantly, methods such as unsupervised and semi-supervised learning have gained quite the attention [7]. These methods often rely on learning normal behavior patterns and flagging deviations from those patterns as anomalies. For time-series data, capturing long-term dependencies is crucial, which has led to the adoption of deep learning models such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) [21], as these models are better suited for handling temporal dependencies than traditional machine learning algorithms.

In addition to deep learning models, ensemble techniques like Isolation Forests [20] and Random Forests have been widely applied to improve anomaly detection performance in high-dimensional datasets. These techniques have shown promise in large-scale infrastructures, where data volumes are high and complex. Furthermore, hybrid approaches, such as combining convolutional neural networks (CNNs) with recurrent networks, have been used to improve the robustness and accuracy of anomaly detection [14].

2.2 AI-Driven Anomaly Detection in CMSWEB Services

The CMSWEB services, which host critical web applications for the CMS experiment at CERN, introduce specific challenges for anomaly detection due to the large-scale telemetry data generated by these services. AI-driven techniques, particularly those leveraging neural networks, have been used in similar cloud-based environments for anomaly detection, focusing on time-series performance metrics such as CPU usage, memory consumption, and network traffic [23]. These methods use unsupervised learning models to automatically detect anomalous patterns without the need for labeled data.

In this work, we implemented and evaluated six different AI models, each utilizing an autoencoder architecture to detect anomalies in CMSWEB service metrics. Autoencoders compress the input data into a lower-dimensional representation and reconstruct it; instances with high reconstruction error are flagged as anomalies [27]. The six models used are:

- **LSTM Autoencoder:** A recurrent neural network (RNN) that captures long-term temporal dependencies in time-series data [21].
- **CNN Autoencoder:** A model that is convolutional and helps in the detection of spatial patterns in sequential data [30].
- **GRU Autoencoder:** A model that is RNN-based and gated, like the LSTM but the architecture is more efficient. [9].
- **Hybrid CNN-LSTM Autoencoder:** A hybrid model combining CNN's ability to capture local patterns with LSTM's strength in modeling long-term dependencies [1].
- **Hybrid CNN-GRU Autoencoder:** A similar hybrid model that replaces LSTM with GRU, offering a more computationally efficient alternative [1].
- **Autoencoder Fully Connected (AEFC):** An autoencoder that is fully connected and does its functions by compressing and rebuilding the input data through dense layers [13], which worked in the least effective way.

In these models, the **Hybrid CNN-LSTM Autoencoder** shown the highest accuracy in the detection of anomalies, while the **Autoencoder Fully Connected** (AEFC) had the poorest performance, most probably because of the inability to capture the inherent complex temporal dependencies in the CMSWEB service data. The AEFC model due to low performance did not undergo hyperparamter tuning, differing from the other five models which were optimized with the help of hyper-parameter tuning methods that enhanced their anomaly detection capabilities.

2.3 A comparison between existing approaches

This work presents approaches that align with researches on AI-driven anomaly detection which were conducted previously, while addressing the CMSWEB related challenges. Studies conducted earlier, like [14], shown the efficiency of hybrid models for time-series detection of anomalies. However, the work that we have conducted distinctively focuses on uniting architectures that are autoencoder-based with hybrid models, like CNN-GRU and CNN-LSTM, which are not being explored extensively in terms of detection of anomalies related to CMSWEB services.

In addition to that, while already existing research work often centers around the single-model approach, our study provides flexibility by presenting multiple models for detection of anomaly. This allows end users to select the most appropriate model for specific performance metrics or use cases. furthermore the five top-performing models with hyperparameter tuning ensures that the system has the capability to adapt to different conditions and metrics in real-time.

3 Dataset and Infrastructure

A strong dataset infrastructure is required for the CMSWEB anomaly detection system that captures extensive time-series perfromance metrics to monitor service behavior. This part gives an outline of the data collected, preprocessing and monitoring architecture. The metrics like CPU and Memory usage are key metrics, these are collected from the CMS Monitoring infrastructure, particularly customized to the Kubernetes-based CMSWEB cluster. It provides a solid foundation for detection of anomaly in critical web services by including detailed configurations for querying namespaces, containers, and specific performance metrics in the dataset architecture.

3.1 Data Collection

The process of data collection for the anomaly detection system in the CMSWEB involves the procedure to gather performance metrics from the prometheus CMS monit framework [3]. In this section, there will be the outlines for the commands used for data retrieval, configuration file structure, and the steps for the preprocess JSON responses from prometheus.

3.1.1 Namespaces and Containers

The logical partitions within a cluster are called Namespaces in Kuberneter which enable the separation and organization of users, resources and permissions in various environments or projects. [5]. They help in simplifying the management of resources, by letting services to exist simultaneously with no interference and are essential in cloud environments for scaling applications. [5]. Each container runs as an isolated process, enhancing security and resource management within Kubernetes clusters, especially useful for microservices architectures [4].

3.1.1.1 Listing Namespaces To obtain a comprehensive list of namespaces in the Kubernetes environment, namespace metadata was retrieved, and the resulting namespaces are shown in Table 1.

Table 1: List of Kubernetes Namespaces

Namespaces		
auth	couchdb	crab
das	dbs	dmwm
dqm	tzero	wma

3.1.1.2 Listing Containers from the Configuration File

The following table lists all containers extracted from the configuration file:

Table 2: List of Containers from the Configuration File

Container	Container	Container
proxy	logstash	crabserver
cron	das-exporter	das-server
das-mongo	check-metric	dbs-global-r
dbs-global-w	dbs-phys03-r	dbs-phys03-w
dbs2go-global-m	dbs2go-global-r	dbs2go-global-w
dbs2go-phys03-m	dbs2go-phys03-r	k8snodemon
podmanager	ms-monitor	ms-output
ms-pileup	ms-rulecleaner	reqmgr2
reqmgr2-tasks	reqmon	reqmon-tasks
t0reqmon	t0reqmon-tasks	workqueue
autodqm	newdqmgui	cmsamqproxy
cmskv	exitcodes	httpgo
kube-eagle	loki	prometheus
prometheus-adapter	monitor	t0wmadatasvc
wmarchive		

3.1.2 Prometheus Monitoring Data Collection

Prometheus is used to collect performance metrics. Two primary metrics were chosen for this work:

- **CPU Usage:** `eagle_pod_container_resource_usage_cpu_cores`
- **Memory Usage:** `eagle_pod_container_resource_usage_memory_bytes`

3.1.2.1 Querying Metrics from Prometheus

To retrieve all available metric names for the `k8s-prod` environment, the following command was used:

```
curl -s --data-urlencode 'match[]={env="k8s-prod"}' \
http://cms-monitoring-ha1.cern.ch:30428/api/v1/label/__name__/values | jq -r '.data[]'
```

The output from this command provides a comprehensive list of available metrics. A sample of the retrieved metrics is shown in Table 3.

Table 3: Sample Metrics Retrieved from Prometheus

Metric
apache_accesses_total
apache_cpuload
apache_exporter_build_info
apache_scoreboard
apache_sent_kilobytes_total
apache_up

3.1.3 Dynamic Query Construction

The `config.json` file defines the parameters for querying Prometheus. The configuration file structure is presented below.

3.1.3.1 Sample `config.json` File

```
{
    "prometheus_url": "http://cms-monitoring-ha1.cern.ch:30428/prometheus/api/v1/query_range"
    "metrics": [
        "eagle_pod_container_resource_usage_memory_bytes",
        "eagle_pod_container_resource_usage_cpu_cores"
    ],
    "app": "kube-eagle",
    "env": "k8s-prod",
    "step": "15s",
    "combinations": [
        ["crab", "crabserver"],
        ["das", "das-exporter"]
    ],
    "start_date": "2024-09-10T05:00:00Z",
    "end_date": "2024-09-13T14:02:58.443475Z",
    "interval_days": 5,
    "output_dir": "/data/outputCSV",
    "models": [
        "HybridCnnLstm",
        "HybridCnnGru",
        "AeCNN"
    ]
}
```

3.1.4 Constructing a Sample Query

Based on the above configurations, the following query is constructed to retrieve the memory usage data for the `crabserver` container within the `k8s-prod` environment:

```
http://cms-monitoring-ha1.cern.ch:30428/prometheus/api/v1/query_range?query=
eagle_pod_container_resource_usage_memory_bytes{app="kube-eagle", env="k8s-prod",
namespace="crab", container="crabserver"}&start=2024-09-10T05:00:00Z&end=2024-09-13T14:02:58.
```

3.1.4.1 Breaking Down the Query

The query used to retrieve data from Prometheus consists of several components:

- **Base URL:** `http://cms-monitoring-ha1.cern.ch:30428/prometheus/api/v1/query_range` – This is the endpoint for querying Prometheus data over a specified time range.
- **Query Parameter:** `query=eagle_pod_container_resource_usage_memory_bytes{app="kube-eagle", env="k8s-prod", namespace="crab", container="crabserver"}` – This parameter specifies the metric and filters the data based on the application, environment, namespace, and container.
- **Time Range Parameters:** `start=2024-09-10T05:00:00Z&end=2024-09-13T14:02:58.443475Z` – These parameters define the time range for the data.
- **Step Parameter:** `step=15s` – This defines the time step between data points in the time series.

3.2 Data Preprocessing

The response from Prometheus is in JSON format and includes metric values over time. Here is a sample JSON response:


```
{
  "status": "success",
  "data": {
    "resultType": "matrix",
    "result": [
      {
        "metric": {
          "__name__": "eagle_pod_container_resource_usage_memory_bytes",
          "apod": "kube-eagle-66fb4cccc6-9hgz8",
          "app": "kube-eagle",
          "container": "crabserver",
          "env": "k8s-prod",
          "host": "cmsweb-k8s-prodsrv-v-zone-a-2-5ffxu34ncym4-node-0",
          "instance": "10.100.249.17:8080",
          "job": "kubernetes-pods",
          "namespace": "crab",
          "node": "cmsweb-k8s-prodsrv-v-zone-c-hl63w7hi3wyr-node-1",
          "ns": "monitoring",
          "phase": "Running",
          "pod": "crabserver-5b6f4d8d77-v5qrw",
```

```
          "qos": "Burstable"
        },
        "values": [
          [
            1726203600,
            "82399232"
          ],
          [
            1726203615,
            "81371136"
          ]
        ]
      }
    ]
  }
}
```


3.2.0.1 Explanation of the JSON Response

- **status:** Indicates the success of the query.
- **data:** Enclose result data
  - **resultType:** Kinds of results (e.g., `matrix` for presenting the data of time series).
  - **result:** The arrangement of metrics.
    - **metric:** Metadata Information about the metric, consisting of labels like `app`, `container`, `namespace`, etc.
    - **values:** A time series data in which every entry is a pair of timestamp and value

3.2.1 Pre-processing and Saving to CSV

We used the python script to do the pre-processing and data-saving processes. The processing and storage of the data taken from the prometheus in a format that is suitable for analyzing further is ensured by this script. The steps which are key to this process are as follows:

1. **Extract Data:** The first step or the beginning of the script involves the extraction of data from the JSON feedback taken from the prometheus. Every feedback or response consists of time series data. Each response contains time-series data from which the value and timestamp fields are removed. The value is a metric recorded at that moment and the timestamp is unix epoch time representing the moment when the measurement was observed. To gather these values into a list of dictionaries, the script iterates over every result in the JSON response. The dictionaries conform to a single data point and consists of the pod name (‘pod‘), environment (‘env‘), value and timestamp , pod name , timestamp, and value.

2. **Format Data:** Then comes the step of extracted data formatting. Initially ‘timestamp‘ values were provided in unix epoch format, they are then converted into datetime format which are human-readable. It becomes a lot more easier after the conversion to analyze and interpret the data. In addition to that, to ensure all the entries are in numeric, the ‘value‘ column is inspected and converted. If there is any entries which are not numeric in nature are coerced in NaN (not a number), gets dropped from the dataset. The step involves:

    - Making sure that each value in the column for ‘value‘ are numeric in nature and non numeric entries are removed.

3. **Create CSV:** Data is saved into CSV files after preprocessing. On basis of the model used, the kind of data (e.g., ‘cpu‘ or ‘memory‘) file and identifiers that are specific like namespace and container, each CSV files is given a name. Following operations are performed by the script:

    - **Creation fo directory:** If there is not existing directory, it creates one.
    - **Creation of dataframe:** For the efficient manipulation of data, the list of dictionaries is changed into a pandas dataframe.
    - **Sorting and Aggregation:** For the computation of mean values where there exists multiple entries for the same timestamp, the dataframe is sorted by ‘timestamp‘ and aggregated.
    - **Saving the files** Using proper headers, the final dataframe is retained to a CSV file (‘timestamp‘, ‘value‘, and ‘env‘). For ensuring a file ready for analysis, the file is written in a very clean formating, removing any extraneous columns.

    The procedure mentioned above makes sure that the data is precisely preprocessed and retained in a structured format, giving an easy access for retrieval and analysis for the steps in the later on stages in the data processing pipeline.

3.2.2 Configurable Time Range

The file ‘config.json‘ allows you to specify the start and end times for data gathering. The start time is adjusted to 30 days prior for the initial data collected. For continuous data collection, the start time is adjusted to the last 2 hours.

3.2.3 Memory Management

Due to the large volume of data, memory usage is monitored, and garbage collection (‘gc.collect()‘) is used to manage memory efficiently during data collection.

3.2.4 Output and Storage

Processed data is saved incrementally to CSV files in the specified output directory (‘/data/outputCSV‘). File naming follows the format:

```
{model_name}_{metric}_{namespace}_{container}.csv
```

These files are then used for training and evaluating the anomaly detection models.

3.2.5 Continuous Data Collection

The data collection process runs in a loop with a 120-second interval between cycles. This ensures that the anomaly detection system remains up-to-date with the most recent data.

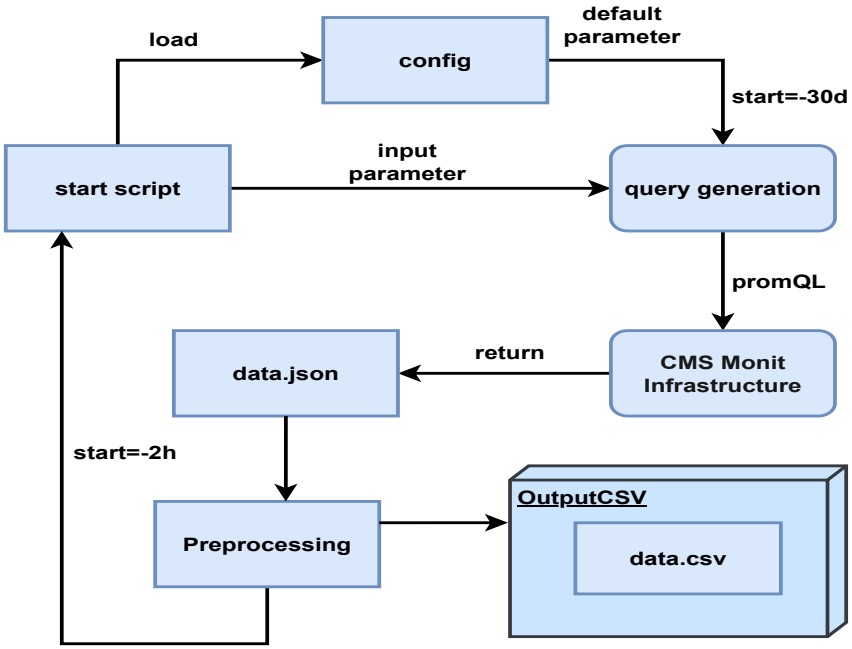


Figure 2: Data Collection Process Overview.

Figure 2 illustrates the data collection workflow. The process begins when the user initiated it through a command. As a result a configuration file with the necessary parameters is triggered, which then gets used to produce a query that is tailored to the parameters in the configuration file. To retrieve data, the query is then executed, which in steps ahead goes through preprocessing and retained in a specific folder with designated name for future use.

4 AI Algorithms for Anomaly Detection in CMSWEB Services

4.1 Autoencoder LSTM

A recurrent neural network (RNN) framework forms the basis of the autoencoder LSTM model, which is particularly designed to obtain temporal dependencies in the time-series data [22]. The impact that past readings have on latest and future observations is basically termed as temporal dependencies [19] [15]. To understand the trends and patterns in the evolving data with time, this concept is very crucial, which allows the models to predict accurately based on the historical behavior [19] [15]. As these services produce performance metrics like CPU and memory usage, which show complex temporal patterns, this is particularly useful in the context of services in CMSWEB.

Architecture: There are two key components of the LSTM-based autoencoder which are the encoder and the decoder [22]. The encoder function by compressing the input time-series data

into a latent representation that is lower-dimensional, on the other hand the decoder functions by reconstructing the original data from the format that was compressed by the encoder [22]. The LSTMs are specifically befitting for this work, because they have the capability to capture both long-term as well as short-term dependencies by maintaining an inner memory state over all the time steps [22]. In this way, it lets the model acquire information and learn the routine behavior patterns in the CMSWEB data over time.

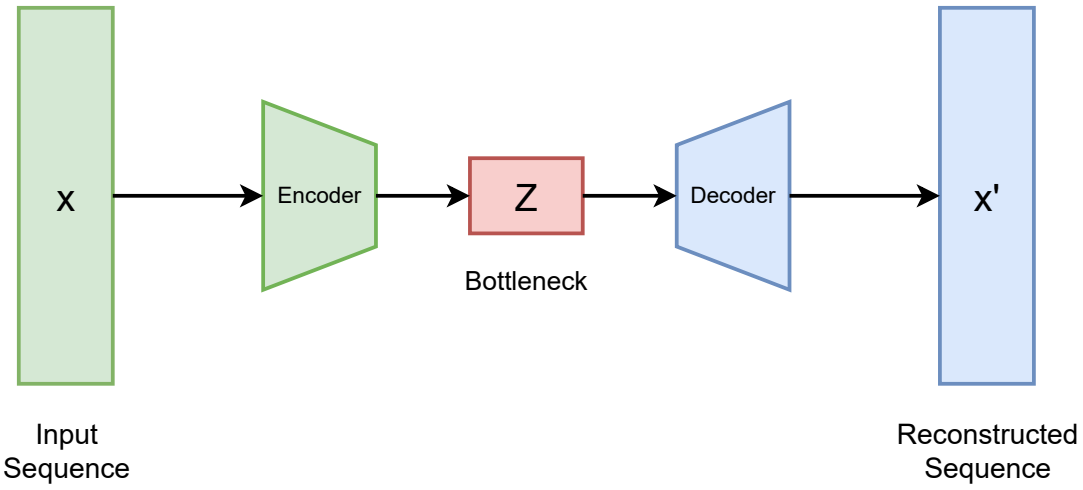


Figure 3: LSTM-Autoencoder Architecture.
removed CC

Figure 3 exhibits the framework of an LSTM autoencoder. The sequence of the input is encoded in small representation which are called the "bottleneck". Then the bottleneck is used by the decoder to rebuild the sequence of the original input.

Advantages: The most vital benefit of LSTMs is their capability to save relevant information over sequences that are long, this advantage in time series analysis is very crucial [22]. With the help of the learning that LSTMs get from the model normal behavior over longer time durations, it can detect indistinct deviations effectively that might be an indication of an anomaly [22]. For the distinct detection between normal fluctuations and the real ones in service performance, the ability of the LSTMs to remember past information that is relevant and discard irrelevant data is very crucial.

Application: With historical data like CPU and memory usage the autoencoder LSTM is trained which is used for detection of anomalies in the CMSWEB services. During training, the model learns to minimize the reconstruction error between the original and the reconstructed data, capturing normal behavior patterns. When exposed to new data, any data points with a high reconstruction error are flagged as potential anomalies, indicating deviations from normal service behavior [26]. For example, a sudden, unexplained spike in CPU usage that the model cannot reconstruct accurately could be flagged as an anomaly.

4.2 Autoencoder CNN

A convolutional neural network (CNN) framework is used by the autoencoder CNN model, which is traditionally has a function in processing images but it also has proven to be effective in identifying spatial patterns in time-series data [29]. Spatial patterns refer to the arrangement of data points in a

specific geometric configuration over time, revealing trends or behaviors across different dimensions [18].

Architecture: In this model, the CNN layers act as feature extractors by applying convolutional filters to the input time-series data, detecting short-term local patterns [29]. With the model that is LSTM-based, there is an encoder-decoder structure in the CNN-based autoencoder, where the input data is compressed by the encoder in a latent space representation, while the reconstruction of the data from this representation is done by the decoder [22] [29].

Advantages: In recognizing patterns that are localized in the data like periodic spikes or anomalies that are isolated which appear in particular time durations, the CNNs are specifically adept [29]. The efficiency of the architecture to capture high dimensional structures of data makes it befitting in the processing of complex, short-term pattern of performance metrics.

Application: The CNN autoencoder in the CMSWEB is trained and prepared on time-series data to learn normal behavior. After it is thoroughly trained, any point of the data that has error in high reconstruction is measured as an anomaly because it is a deviation from the patterns that were learned [22]. For example, an abrupt increase in the latency of the network which does not align with the historical patterns could be detected by the model as an anomaly.

4.3 Autoencoder Fully-Connected and GRU Models

Comparative to the other frameworks, the fully-connected autoencoder (AEFC) model had the limited effectiveness in the detection of anomaly. However the model which offered more efficient alternate to the LSTM was the gated recurrent unit (GRU) and it was a more suitable option to capture the temporal dependencies in the data of time-series.

GRU (Gated Recurrent Unit): GRUs are the kind of RNN models that has a simple architecture as compared to the LSTMs [25]. They reduce the computational overhead in addition to maintaining their ability to model sequential dependencies with the process of uniting the forget and input gates of an LSTM in to a single update gate. [25]. Autoencoders which are GRU-based has a similar operation like the LSTM-based models, that learn the compressing and reconstructing time-series data to anomaly detection which are based on the reconstruction errors [25].

Fully-Connected (AEFC): The model that is composed solely of layers that are fully-connected are the AEFC models, they struggle to obtain the temporal relations which are inherent in the time-series data. It has a lower performance in the detection of anomalies because it is unable to recognize sequential dependencies, which are evidenced in this study. Resultanly, this model due to its poor results was not subjected to the hyperparameter tuning.

5 Hybrid CNN-LSTM and CNN-GRU Autoencoders

In the work that we have done, we use hybrid models uniting CNN with LSTM and layers of GRU. This framework are suitable for detection of time-series anomaly, as they have the capability to catch both long-term dependencies and the localized patterns in the data.

Architecture: There are two main components of the hybrid models. The CNN component that processes the time-series dat input by employing convolutional layers which then detect the short-term features like local fluctuations in the service metrics [16]. Then these features gets passed to the

LSTM or GRU layers, which in a sequence process the data and then long-term temporal dependencies are modeled [16]. This union helps the models to detect anomalies that are short-term like spikes in the CPU usage which are sudden and it also helps in determining the long-term trends, like the gradual memory degradation [16].

Advantages:

- **CNN**: The layers of CNN specifically adapt in identifying short-term patterns and anomalies that are local which might occur in specific windows of time, for example, surges in the network traffic which are sudden [30].
- **LSTM/GRU**: Contrarily the layers of GRU or LSTM, work at modeling the dependencies in the data that are long-term. This is a crucial feature as in the detection of the slow and gradually developing anomalies over time, like the slow increases in the resource usages [22] [25].

Application in CMSWEB: Due to the ability of these hybrid models to capture wide range of anomalies, these hybrid models are very beneficial in the CMSWEB services. The CNN layers has the ability to identify localized variations, while the GRU or LSTM layers can monitor trends that are long-term. For example, an abrupt peak in the CPU usage, with a sustained higher utilization for prolonged durations, could be an indication of an issue in the system's performance. With the strengths of LSTMs/GRU and CNNs, this complementary combination leverages the functions of these hybrid models and provide a robust detection of anomalies in the CMSWEB services, making sure short-term as well as long-term anomalies are efficiently detected.

5.1 Performance Comparison

Across two usage types that is CPU and Memory, the performance of six implemented models is evaluated. We utilized eight applications, with each application generating datasets for both CPU and memory usage, resulting in a total of sixteen datasets. For simplicity and clarity, the performance comparison in this section presents the average values across all datasets for each model. This aggregated view allows us to make general observations about model performance in terms of error metrics, anomaly thresholds, and time metrics.

5.1.1 Error Metrics

The error metrics (RMSE, MAE, MASE, R^2) provide insights into how well each model can reconstruct normal behavior. These metrics are calculated for both CPU and memory-based applications, but here we present the average values across all datasets on training data.

- **RMSE (Root Mean Square Error)**: The square root of the average squared differences between predicted and actual values [6] [8].

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of observations.

- **MAE (Mean Absolute Error)**: The average absolute difference between the predicted and actual values [6] [8].

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **MASE (Mean Absolute Scaled Error)**: A scaled version of MAE to account for data scale [24] [8].

$$\text{MASE} = \frac{\text{MAE}}{\frac{1}{n}\sum_{i=1}^{n}|y_i - \bar{y}|}$$

where $\bar{y}$ is the mean of the actual values.

- R^2 **(Coefficient of Determination)**: The proportion of variance in the data explained by the model [8].

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

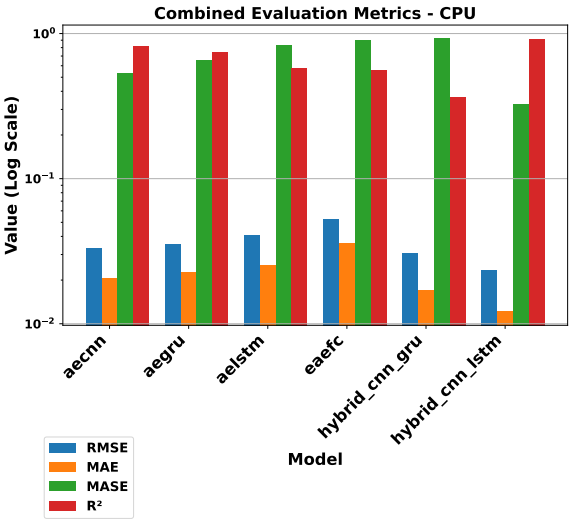


Figure 4: Comparison of error metrics for CPU-based applications on training data.

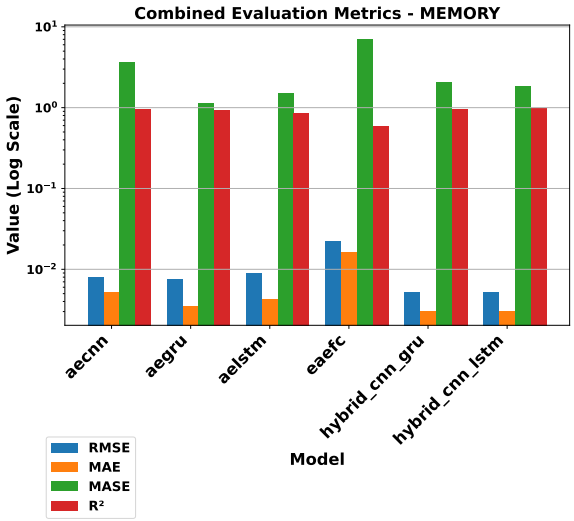


Figure 5: Comparison of error metrics for memory-based applications on training data.

Figures 4 and 5 present the average values of RMSE, R^2, MASE, and MAE for all models across various applications on two key metrics: CPU and memory usage. To enable visual comparison, the error values are represented on a logarithmic scale. This adjustment helps normalize the range of different error metrics, allowing for a clearer comparison of model performance across models.

5.1.2 Anomaly Thresholds

The thresholds used to assess anomaly detection are calculated as follows:

- **Mean + 1.5 Standard Deviation (std):**

$$\text{Threshold}_1 = \mu + 1.5\sigma$$

where μ is the mean of the data and σ is the standard deviation.

- **95th Percentile:**

$$\text{Threshold}_2 = P_{95}$$

where P_{95} represents the 95th percentile value of the data distribution.

- **99th Percentile:**

$$\text{Threshold}_3 = P_{99}$$

where P_{99} is the 99th percentile value of the data distribution.

- **Median Absolute Deviation (MAD):**

$$\text{Threshold}_4 = \text{median}(|X_i - \text{median}(X)|)$$

where X is the dataset and median(X) is the median value.

5.1.2.1 Percentage Above Threshold The percentage above threshold measures the proportion of data points that exceed predefined thresholds. This gives an indication of how often the model signals an anomaly, with the thresholds chosen to capture extreme deviations.

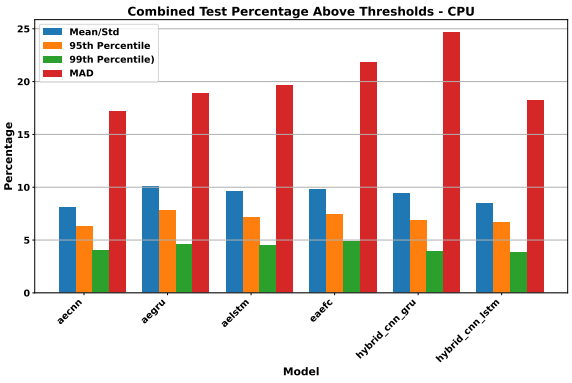


Figure 6: Percentage above threshold for CPU-based applications.

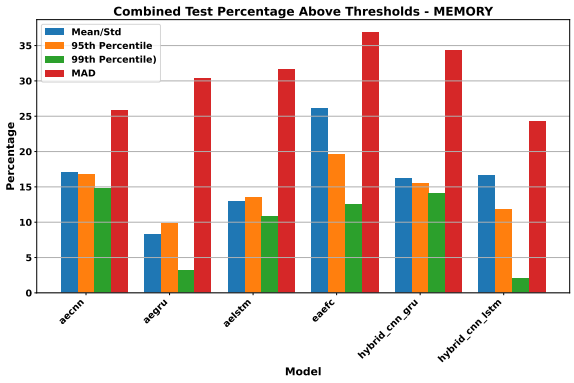


Figure 7: Percentage above threshold for memory-based applications.

Figures 6 and 7 illustrate the percentage of data points in the testing dataset that each model identified as anomalous. The results are presented separately for CPU and memory metrics, allowing for a clear visual comparison of model sensitivity to anomalies across these two key resource indicators.

5.1.2.2 Mean Above Threshold The mean above threshold measures the average magnitude by which predictions exceed the threshold. It indicates the severity of the anomalies detected by each model.

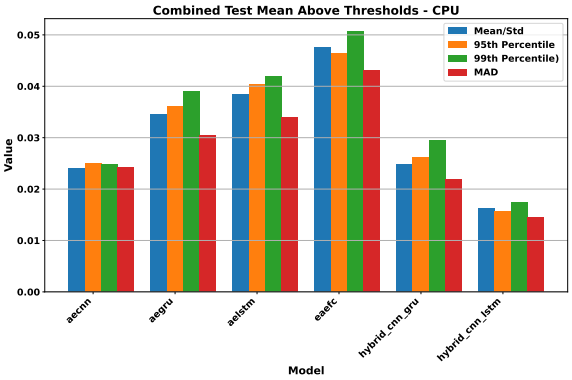


Figure 8: Mean above threshold for CPU-based applications.

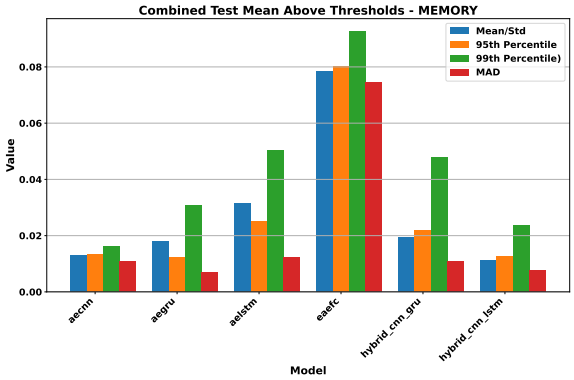


Figure 9: Mean above threshold for memory-based applications.

Figures 8 and 9 illustrate the mean above threshold values for CPU- and memory-based applications. These values represent the average extent by which predicted data points exceed the anomaly threshold. Higher mean above threshold values indicate more severe anomalies detected by each model, providing insight into the degree of deviation in model predictions across different application resources.

5.1.3 Time Metrics (Training and Inference Time)

Efficiency in terms of time is critical for real-time anomaly detection. We evaluated the models based on their training time and inference time, averaged over the CPU and memory-based applications. Training time reflects the computational cost during model training, while inference time is important for real-time anomaly detection.

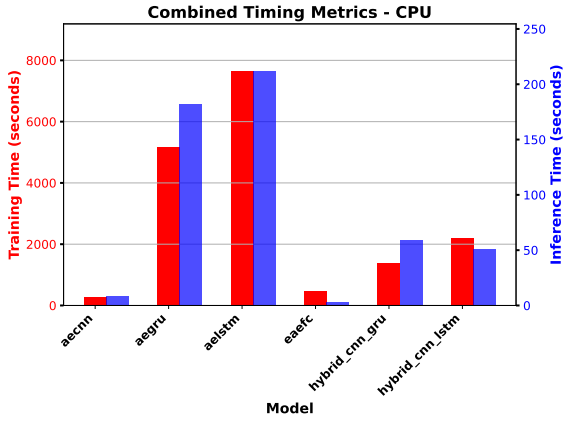


Figure 10: Training and inference times for CPU-based applications.

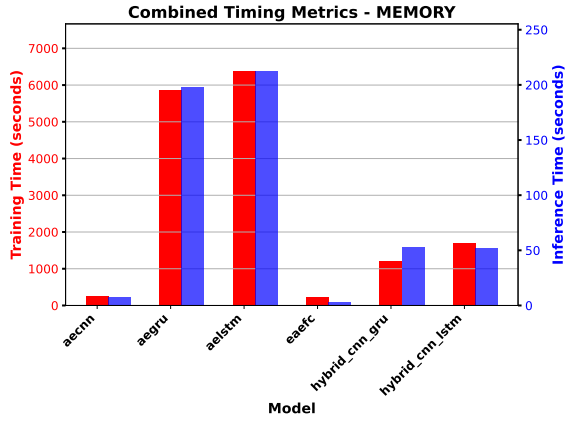


Figure 11: Training and inference times for memory-based applications.

5.1.4 Summary of Model Comparison

For drawing a comparison between the performances of the different models, we assess them on multiple criteria like training and testing metrics of error, detection of anomaly thresholds and time efficiency. By using a dataset of the obtained results from training the models on memory and

CPU usage in the, this comparison was conducted. The following method outlines the process of comparison:

1. **Error Metrics:** RMSE, MAE, MASE and (R^2) are included as error metrics for the training as well as the testing stages The above error metrics give a comprehensive demonstration of how good during the training and inference, each model fits the data.

2. **Anomaly Thresholds:** Based on measures of statistics like mean + standard deviation, 95th percentile, 99th percentile and MAD, multiple anomaly detection thresholds were defined. The mean of the exceeding data point from the thresholds and the percentage of the data points above the thresholds were assessed for training as well as testing phases.

3. **Time Efficiency:** The evaluations of the models were also done on the basis of their efficiency in computation. Particularly, our consideration was the time needed for training and inferences, which is an important aspect for the determination of the applicability of the models in real-time in an environment that is operational.

4. **Normalization and Ranking:** The results for all the metrics were normalized with the help of MinMaxScaler to provide a fair comparison, this ensured the equal contribution of each metric to the final model ranking. To provide a fair comparison, the results across all metrics were normalized using MinMaxScaler, ensuring that each metric contributed equally to the final model ranking. To maintain the consistency in the process of ranking the metrics where higher values are better (e.g., R^2),the normalized values were inverted.

5. **Weighted Scoring:** Using a weighted average of the error metrics, thresholds of anomaly and time efficiency, the final model score was calculated. The model which was final was computed by using balanced weighting of thresholds of anomaly, error metrics and time efficiency. There was weightings of 25% for training error, 25% for testing error, 25% for anomaly threshold performance, and 25% for time efficiency were applied . The scores which were weighted was summed up to form an overall score, that was utilized for the ranking of models.

We calculate each models‘s average scores, normalizes the outcomes, and models are ranked accordingly. The summary of the comparison for the results are in table 4, that exhibits that the *Hybrid CNN-LSTM* model got best performance results in across all criteria of evaluation, with the least error metrics, best detection performance of anomalies, and a very reasonable time efficiency aspect. Contrary to that, the *Autoencoder Fully-Connected* (AEFC) had the worst performance, with a very high error metric, which makes it the less suitable model for anomaly detection in real-time.

Model	Final Score	Rank
Hybrid CNN-LSTM	0.087	1
AECNN	0.178	2
Hybrid CNN-GRU	0.275	3
AEGRU	0.371	4
AELSTM	0.528	5
AEFC	0.704	6

Table 4: Model Performance Ranking Based on Final Score

In a short summary, the *Hybrid CNN-LSTM* model exhibited high execution in terms of accuracy as well as efficiency, which makes it the most suited model for detection of anomalies in real-time in the CMSWEBs services. On the contrary, the *Autoencoder Fully-Connected* is a less appropriate model for practical use in this context because of a higher error.

6 Model Selection and Hyperparameter Tuning

6.1 Model Selection

The process of selecting a model for anomaly detection was a crucial step in getting an optimal performance. Various frameworks like the Traditional autoencoders, recurrent neural networks (RNNs) like GRUs and LSTMs and convolutional neural networks (CNNs) were considered. In these models, the **Hybrid CNN-LSTM model** was selected on the basis of its ability to unit both spatial and temporal feature extraction which makes it a suitable choice for anomaly detection work in the time-series. The component that is CNN functions well at obtaining spatial dependencies through convolutional filters, on the other hand the LSTM component catches the temporal dependencies which are inherently present in the dataset. This approach of hybrid modeling came out to be very effective in dealing the characteristics of the time-series data that was gathered from the CMSWEB services.

6.2 Hyperparameter Tuning

We run a comprehensive hyperparameter tuning procedure using **Random Search** to optimize the performance of hybrid CNN-LSTM model which is a a practical way to effectively and efficiently explore the hyperparameter space. On critical parameters tuning was performed which is summarized in the table 5. The aim was to lessen *val_loss*, the loss of validation, which worked as the optimization metric.

Table 5: Hyperparameter Tuning for Hybrid CNN-LSTM Model

Parameter	Range/Values	Purpose
Filters	64, 128, 192, 256	Vary CNN filters for feature extraction
Kernel Size	2, 3, 4, 5	Adjust receptive field in convolution layers
LSTM Units	64, 128, 192, 256	Control capacity to capture temporal dependencies
Dropout Rate	0.1 to 0.5 (increments of 0.1)	Prevent overfitting
Optimizer	Adam, RMSprop	Tune optimization strategy

Tuning was run of fifty plus epochs with a batch size of thirty-two, and to prevent overfitting early stopping was implemented. It had a significant impact and model‘s generalization was improved, with a less validation loss as compared to the version that was untuned.

6.3 Evaluation Metrics

Using the metrics mentioned below, the before and after hyperparameter performance of the hybrid CNN-LSTM model was evaluated:

- **Losses in Training and Validation**: These metrics calculate the performance of the model during training and help evaluate generalization to new, data which is unseen [12]. *Training Loss* shows how well the model fits the training data, while *Validation Loss* evaluates its accuracy on a separate validation set, representing unseen data [12].

  The loss function that is used is the mean squared error (MSE), that measures the average squared difference the values that are actual (y_i) and predicted ($\hat{y}_i$) [8]:

$$\text{Loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  This calculation of MSE gives an uncomplicated reading of prediction error which is used to enhance tuning of the model and for the prevention of over-fitting [12].

- **Test Data Performance**: The tuned model was evaluated on the test data using the same metrics, to ensure that the model did not overfit the training data and generalizes well to unseen data.

The results of the **training loss** and **validation loss** for both pre- and post-hyperparameter tuning are visually represented in Figure 12. The comparison clearly shows that after tuning, the model's validation loss is significantly lower, indicating better generalization.

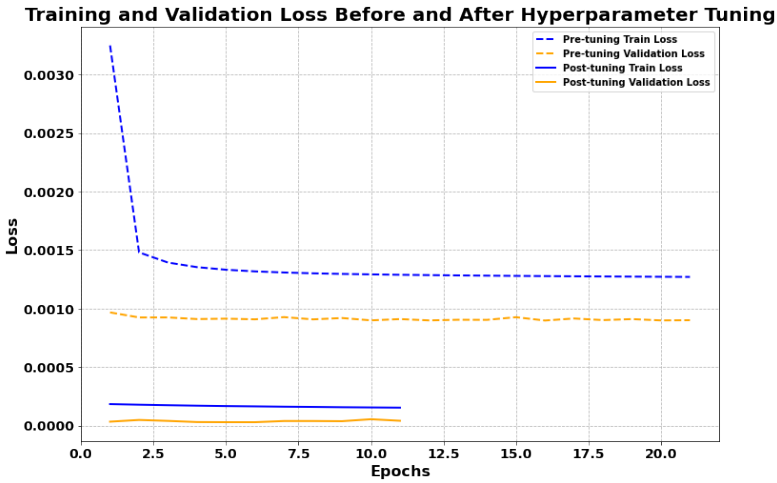


Figure 12: Training and Validation Loss before and after Hyperparameter Tuning

7 AI-based Application Design and Architecture

7.1 System Overview

The proposed AI-based application architecture is designed to monitor critical CMSWEB services in real-time, focusing on anomaly detection to enhance operational efficiency and reliability. The architecture comprises several key components:

- **Data Collection Module:** Utilizes a Python script to fetch live metrics from Prometheus endpoints for various services.

- **Anomaly Detection Engine:** Employs multiple machine learning models, including hybrid architectures and autoencoders, for training on historical data and real-time inference.
- **Alerting System:** A robust alert mechanism that triggers notifications based on detected anomalies.
- **Web Interface:** A user-friendly web application for monitoring service performance and visualizing insights through charts and dashboards.
- **Configuration Management:** A flexible configuration system that allows dynamic adjustments to parameters such as metrics, services, and models used for training and detection.

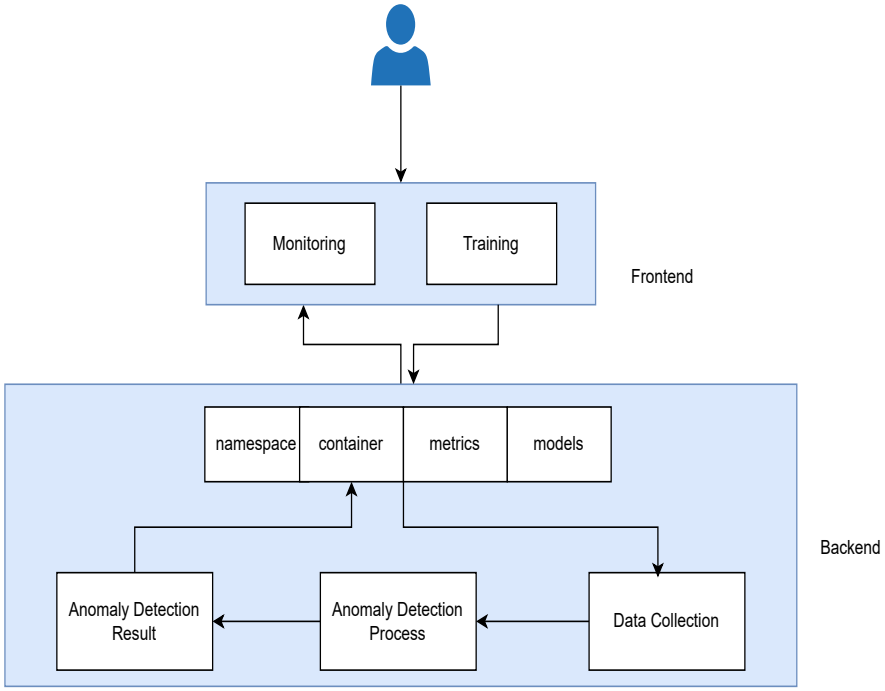


Figure 13: Application Architecture

7.2 Data Flow

The system continuously fetches live data from CMSWEB services through the following process:

1. **Metrics Retrieval:** A script is responsible for executing `curl` commands to query Prometheus for metrics associated with different services and namespaces. These metrics include CPU and memory usage, among others.
2. **Data Processing:** Collected metrics are processed and stored in CSV format in a designated output directory. This structured format enables easy access for subsequent model training.
3. **Data Ingestion:** Every 30 seconds, the application checks for new metrics, ensuring that the data is up-to-date for both training and real-time anomaly detection.

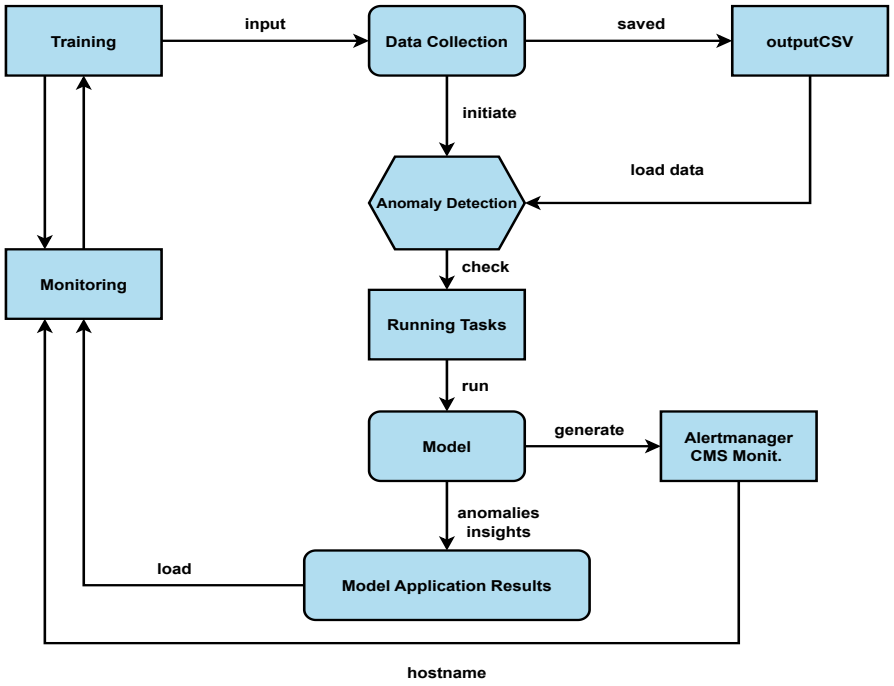


Figure 14: Overview of application workflow

The Figure 14 shows the work flow for overall application. The system first collects training data which is then used to train a model. Once the model is trained, it can be used to detect anomalies in new data. The system also consists of parts that observe the performance of the model and when anomalies are detected, they generate alerts.

7.3 Real-Time Training and Detection

For making sure that the system updates the model with new data continuously and does anomaly detection in real time, the following procedure is applied:

1. **Continuous Learning:** The framework is made for the implementation of the continuous training, where it retrains the models periodically with the help of data that is recently collected from the CMSWEB services. There is one-hour sliding window that it uses for training of recent data, while to establish baseline performance, historical data is leveraged.

2. **Anomaly Detection:** For the inference on the incoming data streams, the models that are trained are deployed. With the arrival of new data points, they are evaluated by the models against the thresholds and learned patterns. Alerts are triggered as there is any deviations from the learned patterns exists, letting the system to identify short-term as well as long-term types of anomalies in the service metrics.

3. **Threshold Management:** Various thresholds (95th percentile, 99th percentile, mean + 1.5 * std and MAD) are formed on the basis of data that is historical for the effective detection of anomalies. Alerts are triggered if there is exceeding data points from these thresholds.

4. **Adaptive Thresholds:** The system in addition to static thresholds, also employs adaptive thresholds for dynamical adjustments based on the past and current data behavior. There are two key components of the adaptive threshold calculation: recent trends and historical performance.ws

   The adaptive threshold is influenced by:

   - *Previous Threshold (T_{prev})* and *Recent Threshold (T_{recent})*: The thresholds of anomaly that were calculated from past and recent data windows, respectively are reflected by these values.
   - *Standard Deviations ($\sigma_{prev}, \sigma_{recent}$)*: Training data's standard deviations from the past and recent windows is utilized to weight the impact of every threshold.
   - *Time Decay Factor*: For giving the recent data more weight, a time decay factor on the basis of the time difference (t_{diff}) between the previous and current data is applied. This factor lowers over time exponentially giving priority to the trends which are recent.

     The adaptive threshold is computed as follows:

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

     where $w = \frac{\sigma_{\text{prev}}}{\sigma_{\text{prev}} + \sigma_{\text{recent}}}$ Illustrates the factor of weighting which is on the basis of standard deviations, and λ is a hyperparameter that controls the rate of time decay.

     This formula makes sure that the adaptive threshold is impacted by the recent as well as past behavior of the data, giving more flexibility and responsiveness to the system for anomaly detection. Resultantly, to evolve the patterns in the performance of CMSWEB service, the system can adapt accordingly and improve the anomaly detection over time as the conditions might change.

7.4 Web Interface and Visualization

The interface of the web is structured for easy use and gives valuable insights to the users about the performance of various CMSWEB services:

1. **Dynamic Dashboards:** Django was used to built, the web application updates dynamically to show the metrics of latest data and results of anomaly detection. Metrics can be filtered by the users on the basis of various parameters like service, environment and application.

2. **Visual Representation:** Visualizations such as pie charts, bar charts and time series plots are included in the application that demonstrate metrics of service performance and the frequencies of anomalies. It makes it easy for the administrators to assess the health of the CMSWEB infrastructure quickly.

3. **User Interaction:** The interface let the users to choose various services and display corresponding metrics which makes it easy to identify potential issues and observe performance over the course of time.

7.5 Alert System

The mechanism of alert is a crucial aspect of the framework. It is made to improve efficiency of the operations by swiftly alerting the stakeholders about the detected anomalies:

1. **Alert Criteria:** To trigger the alerts on the basis of defined thresholds for various metrics, logic is implemented by the alert module to identify the alert timing. On the basis of the level of anomaly, the alerts can specify varying levels of severity (critical, warning).

2. **Notification Channels:** Alerts are sent to multiple Alert manager endpoints to ensure reliability and redundancy. This multi-endpoint approach ensures that notifications reach the intended recipients even if one of the services is temporarily unavailable.

3. **Operational Impact:** By providing timely alerts, the system enhances the ability of administrators to respond to potential threats or performance issues, thereby minimizing downtime and maintaining the reliability of critical services in the CMSWEB cluster.

8 Results and Discussion

8.1 Performance Results

The performance of the models was evaluated using various error metrics, and the Hybrid CNN-LSTM model emerged as the best-performing model. This model demonstrated superior results across key metrics, including RMSE, MAE, and R-squared for both training and testing datasets.Figure 15 and Figure 16 provides a graphical comparison of the Error Metrics values for test data across all models. This visualization clearly highlights the superior performance of the Hybrid CNN-LSTM model, with the lowest RMSE and MAE, compared to the other models.

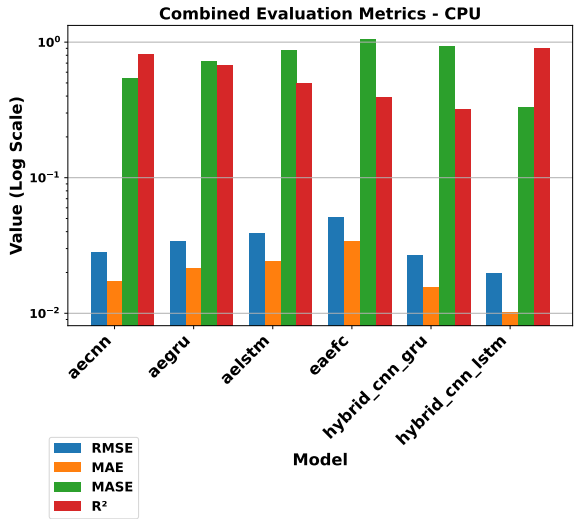


Figure 15: Error Metrics for CPU-based applications - Testing data.

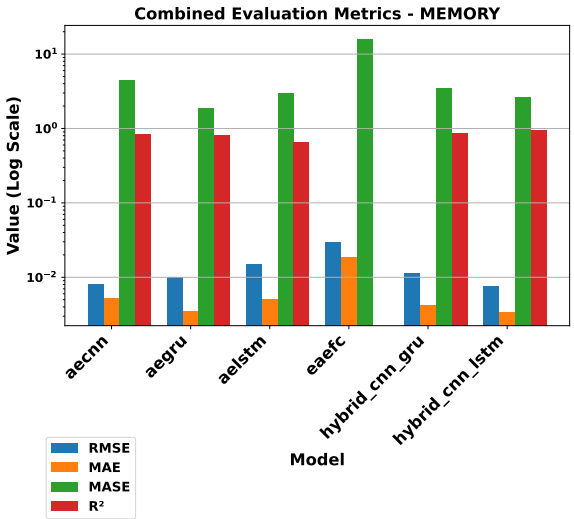


Figure 16: Error Metrics for Memory-based applications - Testing data.

8.2 Visualization and Web Insights

The application that is based web gives a comprehensive visualization of service performance over time. In addition to the alerts to the administrators, it also provides historical insights which are helpful in proactive monitoring. Furthermore, The addition of pie charts and bar charts gives a holistic display of health of the system by summarizing the percentage of exceeding data points from the anomaly thresholds.

8.3 Impact on CERN Operations

This anomaly detection system with its implementation has the potential to increase the monitoring process at CERN to a great extent. With delivering the alerts in real-time in important services such as WMCore and DBS, the system makes sure that any degradation in the performance is flagged immediately, which allows for a swift intervention. In this way, this reduces the possibility of longer downtime in the service, Which forms a more stable environment for the operations. In addition to it, the models adapt to the changing service behaviors because of the continuous learning approach, making sure that the system keeps its relevancy even with the growth and changes in the CMSWEB infrastructure.

9 Conclusion

9.1 Summary of Findings

The focus of this research project is to develop and implement the techniques of anomaly detection in the web services of the CMSWEB framework at CERN. Several models of deep learning were assessed in terms of their effectiveness in anomaly detection through the extraction of performance data from the infrastructure of CMS monitoring. Between these tested models, CNN-LSTM hybrid model shown a superior performance, obtaining the least root mean square error (RMSE) and mean absolute error (MAE) on training as well as test datasets. The ability of this model to generalize well shows that it is robust and well-suited for monitoring application in real-time in the high-performance environments.

9.2 Future Work

Looking forward, there are number of opportunities that exist for improving the current system. The improvements that can occur in the future might include:

- **Exploration of Additional Algorithms:** Looking in to other machine learning and algorithms of deep learning could give an idea about the potential improvements in performance and robustness.
- **Scalability Enhancements:** Accommodating increased volumes of data and broad range of services by scaling of the application is going to be crucial for deployment scenarios in the future.
- **Integration of Predictive Maintenance Features:** To enable the systems to predict possible failures of service, an addition of predictive maintenance feature would help. The system can

forecast the potential failures according to the historical patterns of anomaly and it can bring in overall improvement in service reliability.

9.3 Implications

The implementation of an effective anomaly detection system has significant implications for high-performance infrastructures like CERN. By leveraging advanced machine learning techniques, the system can provide real-time insights into service performance, enabling proactive management of critical web services. This capability not only enhances operational efficiency but also contributes to the reliability and stability of complex systems. As such, continuous improvements in anomaly detection methodologies will play a pivotal role in the evolving landscape of high-performance computing environments.

References

[1] Mobarak Abumohsen et al. “Hybrid machine learning model combining of CNN-LSTM-RF for time series forecasting of Solar Power Generation”. In: *e-Prime - Advances in Electrical Engineering, Electronics and Energy* 9 (2024), p. 100636. ISSN: 2772-6711. DOI: https://doi.org/10.1016/j.prime.2024.100636. URL: https://www.sciencedirect.com/science/article/pii/S277267112400216X.

[2] Aamir Ali et al. “Implementation of New Security Features in CMSWEB Kubernetes Cluster at CERN”. In: *EPJ Web of Conferences*. Vol. 295. EDP Sciences. 2024, p. 07026.

[3] Christian Ariza-Porras, Valentin Kuznetsov, and Federica Legger. “The CMS monitoring infrastructure and applications”. In: *Computing and Software for Big Science* 5.1 (Jan. 2021), p. 5. ISSN: 2510-2044. DOI: 10.1007/s41781-020-00051-x. URL: https://doi.org/10.1007/s41781-020-00051-x.

[4] Kubernetes Authors. *Containers | Kubernetes Documentation*. https://kubernetes.io/docs/concepts/containers/. 2024.

[5] Kubernetes Authors. *Namespaces | Kubernetes Documentation*. https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/. 2024.

[6] Alexei Botchkarev. “A new typology design of performance metrics to measure errors in machine learning regression algorithms”. In: *Interdisciplinary Journal of Information, Knowledge, and Management* 14 (2019), pp. 045–076.

[7] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly detection: A survey”. In: *ACM Comput. Surv.* 41.3 (July 2009). ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. URL: https://doi.org/10.1145/1541880.1541882.

[8] Davide Chicco, Matthijs J Warrens, and Giuseppe Jurman. “The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation”. In: *Peerj computer science* 7 (2021), e623.

[9] Kyunghyun Cho. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).

[10] Cms Collaboration et al. “The CMS experiment at the CERN LHC”. In: *Journal of instrumentation* 3.August 2008 (2008), pp. 1–334.

[11] Domenico Giordano et al. “Anomaly detection in the CERN cloud infrastructure”. In: *EPJ Web of Conferences*. Vol. 251. EDP Sciences. 2021, p. 02011.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[13] Faming Huang et al. “A deep learning algorithm using a fully connected sparse autoencoder neural network for landslide susceptibility prediction”. In: *Landslides* 17 (2020), pp. 217–229.

[14] Kyle Hundman et al. “Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 387–395.

[15] RJ Hyndman. *Forecasting: principles and practice*. OTexts, 2018.

[16] M.S. Ibrahim, S.M. Gharghory, and H.A. Kamal. "A hybrid model of CNN and LSTM autoencoder-based short-term PV power generation forecasting". In: *Electrical Engineering* 106 (2024), pp. 4239–4255. DOI: 10.1007/s00202-023-02220-8.

[17] Muhammad Imran et al. "Migration of CMSWEB cluster at CERN to Kubernetes: a comprehensive study". In: *Cluster Computing* 24.4 (2021), pp. 3085–3099.

[18] Zahra Khademi, Farideh Ebrahimi, and Hussain Montazery Kordy. "A transfer learning-based CNN and LSTM hybrid deep learning model to classify motor imagery EEG signals". In: *Computers in Biology and Medicine* 143 (2022), p. 105288. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2022.105288. URL: https://www.sciencedirect.com/science/article/pii/S0010482522000804.

[19] G. Kitagawa. *Introduction to Time Series Modeling*. 1st. Chapman and Hall/CRC, 2010. DOI: 10.1201/9781584889229.

[20] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation-based anomaly detection". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6.1 (2012), pp. 1–39.

[21] Pankaj Malhotra et al. "Long short term memory networks for anomaly detection in time series." In: *Esann*. Vol. 2015. 2015, p. 89.

[22] H.D. Nguyen et al. "Forecasting and Anomaly Detection approaches using LSTM and LSTM Autoencoder techniques with the applications in supply chain management". In: *International Journal of Information Management* 57 (2021), p. 102282. ISSN: 0268-4012. DOI: https://doi.org/10.1016/j.ijinfomgt.2020.102282. URL: https://www.sciencedirect.com/science/article/pii/S026840122031481X.

[23] Favour Olaoye and Kaledio Potter. "AI-Driven Anomaly Detection in Critical Infrastructure". In: ().

[24] Patrick Omoumi et al. "To buy or not to buy—evaluating commercial AI solutions in radiology (the ECLAIR guidelines)". In: *European radiology* 31 (2021), pp. 3786–3796.

[25] Zhaowei Qu et al. "A Unsupervised Learning Method of Anomaly Detection Using GRU". In: *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*. 2018, pp. 685–688. DOI: 10.1109/BigComp.2018.00126.

[26] Mahmoud Said Elsayed et al. "Network Anomaly Detection Using LSTM Based Autoencoder". In: *Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks*. Q2SWinet '20. Alicante, Spain: Association for Computing Machinery, 2020, pp. 37–45. ISBN: 9781450381208. DOI: 10.1145/3416013.3426457. URL: https://doi.org/10.1145/3416013.3426457.

[27] Mahmoud Said Elsayed et al. "Network anomaly detection using LSTM based autoencoder". In: *Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks*. 2020, pp. 37–45.

[28] Hyunsun Song and Hyunjun Choi. "Forecasting stock market indices using the recurrent neural network based hybrid models: CNN-LSTM, GRU-CNN, and ensemble models". In: *Applied Sciences* 13.7 (2023), p. 4644.

[29] Chunyong Yin et al. “Anomaly Detection Based on Convolutional Recurrent Autoencoder for IoT Time Series”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52.1 (2022), pp. 112–122. DOI: 10.1109/TSMC.2020.2968516.

[30] Bendong Zhao et al. “Convolutional neural networks for time series classification”. In: *Journal of systems engineering and electronics* 28.1 (2017), pp. 162–169.