Time Series Alarm Forecasting

Thomas Stoeckl

2026-02-27

Table of contents

1 Overview

Alarm management is a critical aspect of industrial automation and control systems. It involves the systematic handling of alarms generated by various processes and equipment to ensure timely response and resolution of issues. Effective alarm management helps in minimizing downtime, improving safety, and enhancing overall operational efficiency. In this project, we focus on forecasting time series data related to alarms, which can help in predicting future alarm occurrences and trends. By leveraging advanced forecasting techniques, we aim to provide insights that can aid in proactive maintenance and decision-making.

In Section 2, we describe the forecasting models, Section 3 covers data preparation and exploration, providing insight into the dataset. Section 4 presents the forecasting results, and Section 5 closes with key findings and future directions.

2 Models

In this section, we provide a concise overview of the four primary models employed for time series forecasting in this project: SARIMAX, Extreme Gradient Boosting (XGBoost), Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM). Each model is briefly described along with its mathematical formulation to elucidate its underlying principles and mechanisms.

2.1 SARIMAX

ARIMA [1] is described as a statistical approach for time-series forecasting under random errors, structured around three components: autoregression (AR), integration (I) , and moving average (MA). Its order is denoted by (p, d, q), where p is the number of AR lags, d is the differencing order, and q is the MA lag.

SARIMAX is an extension of the ARIMA family that includes both seasonal dynamics and exogenous variables to improve forecasting performance [2].

Seasonal dynamics are incorporated via the seasonal order $(P, D, Q)^s$ in addition to the non-seasonal order (p, d, q), where s is the number of time steps per season. Here, P is the seasonal AR lag, D is the seasonal differencing order, and Q is the seasonal MA lag. The backshift operator is denoted by G, the observed series by x_t, and the prediction error by e_t.

$$\phi_p(G)\phi_P(G^s)(1-G)^d(1-G^s)^D x_t = \gamma_q(G)\omega_Q(G^s)e_t$$

In this representation, $(1-G)^d$ is the non-seasonal differencing operator and $(1-G^s)^D$ is the seasonal differencing operator. The polynomials $\phi_p(G)$ and $\gamma_q(G)$ capture non-seasonal AR and MA behavior, while $\phi_P(G^s)$ and $\omega_Q(G^s)$ capture seasonal AR and seasonal MA behavior.

The characteristic polynomials are specified as follows.

$$AR : \phi_p(G) = 1 - \phi_1 G - \phi_2 G^2 - \cdots - \phi_p G^p$$

$$MA : \gamma_q(G) = 1 - \gamma_1 G - \gamma_2 G^2 - \cdots - \gamma_q G^q$$

$$SAR : \phi_P(G^s) = 1 - \phi_1 G^s - \phi_2 G^{2s} - \cdots - \phi_P G^{Ps}$$

$$SMA : \omega_Q(G^s) = 1 - \omega_1 G^s - \omega_2 G^{2s} - \cdots - \omega_Q G^{Qs}$$

SARIMAX extends SARIMA by adding exogenous variables, which are external features that enhance predictive accuracy. These inputs can be modeled as parallel time series correlated with the target series. SARIMAX can be expressed as:

$$\phi_p(G)\phi_P(G^s)(1-G)^d(1-G^s)^D x_t = \sum_k \alpha_k y_{k,t} + \gamma_q(G)\omega_Q(G^s)e_t$$

In this formulation, k indexes the exogenous variables. $y_{k,t}$ denotes the value of the k-th exogenous factor at time t, and α_k is the coefficient associated with that factor.

2.2 XGBoost

XGBoost (Extreme Gradient Boosting) is a scalable gradient boosting framework based on decision trees. It constructs an ensemble of regression trees to approximate complex nonlinear relationships and is widely used due to its strong empirical performance and computational efficiency [3].

Given data $\{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$, where n is the number of observations and p is the number of features, XGBoost models the prediction as an additive ensemble of trees:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i),$$

where each f_k is a regression tree mapping an input x_i to a leaf index and associated leaf weight. The ensemble is learned by minimizing the following objective:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} L(\hat{y}_i, y_i) + \sum_{k=1}^{K} \Omega(f_k),$$

where L is a differentiable loss (e.g., squared error, logistic loss) and

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

penalizes the number of leaves T and the magnitude of leaf weights w_j to control model complexity. γ and λ are hyperparameters controlling the strength of regularization,

XGBoost employs stagewise additive training. Let $\hat{y}_i^{(t)}$ denote the prediction for instance i at iteration t. At step t, we need function f_t that minimizes

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t).$$

The objective restricted to the new tree is approximated by a second-order Taylor expansion of the loss around $\hat{y}_i^{(t-1)}$. Defining first- and second-order derivatives

$$g_i = \frac{\partial}{\partial \hat{y}_i^{(t-1)}} L(y_i, \hat{y}_i^{(t-1)}), \qquad h_i = \frac{\partial^2}{\partial (\hat{y}_i^{(t-1)})^2} L(y_i, \hat{y}_i^{(t-1)}),$$

the approximate objective for tree f_t decomposes over leaves. If I_j denotes the set of samples assigned to leaf j, the optimal leaf weight for leaf j is

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda},$$

and the corresponding score of a fixed tree structure is

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T.$$

This scoring function is used to greedily construct trees: for a candidate split that partitions a node's index set I into I_L and I_R, the loss reduction is

$$\mathcal{L}_{\text{split}} = \frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma.$$

2.3 RNN

A dynamical system is recursive and its classical form is given by

$$h_t = f_\theta(h_{t-1}, x_t).$$

Here, t denotes the time step, h_t is the hidden state at time t, and $f_\theta(\cdot)$ is a function fixed between the states of all time steps. With external inputs, x_t denotes the input signal at time t [4].

The hidden state can be viewed as a summary of the past sequence of inputs and states. If a different function f_θ is used for each sequence length, the model cannot generalize. Generalization across sequences of arbitrary length requires sharing the same parameters at every time step and across all states. A dynamical system with such parameter sharing can be implemented as a neural network with weights; this is called a RNN (Recurrent Neural Network).

Let $x_t \in \mathbb{R}^d$ denote the input, $h_t \in \mathbb{R}^p$ the hidden state, and $y_t \in \mathbb{R}^q$ the output at time step t. Here, d is the input dimension (number of features), p is the hidden-state dimension, and q is the output dimension. Let $W \in \mathbb{R}^{p \times p}$ be the weight matrix between states, $U \in \mathbb{R}^{p \times d}$ the weight matrix between the inputs and states, and $V \in \mathbb{R}^{q \times p}$ the weight matrix between the states and outputs. The bias terms for the state and output are denoted by $b_i \in \mathbb{R}^p$ and $b_y \in \mathbb{R}^q$, respectively. A standard RNN, as shown in Fig. 1, is defined by:

$$h_t = \tanh(Wh_{t-1} + Ux_t + b_i)$$
$$y_t = Vh_t + b_y$$

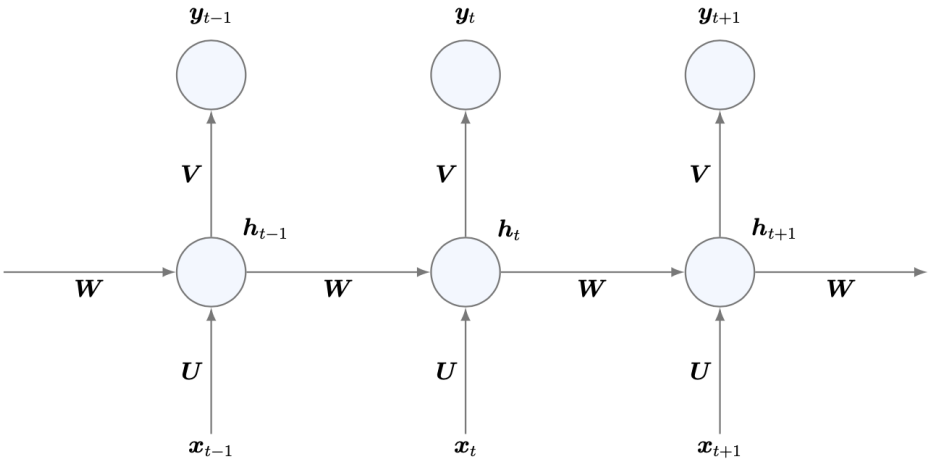

Figure 1: RNN

Training uses backpropagation through time. Because gradients are products across many time steps, long-term dependencies can cause vanishing or exploding gradients.

2.4 LSTM

LSTM (Long Short-Term Memory) [5] networks are a specialized class of RNNs designed to address the problem of learning long-term dependencies in sequential data. Unlike standard RNNs, LSTMs learn from the data itself when information should be retained over long time spans and when it should be discarded. This adaptive memory behavior allows LSTMs to model both short-term and long-term temporal dependencies effectively.

An LSTM processes a sequence of cells across time steps. This is illustrated in Fig. 2.

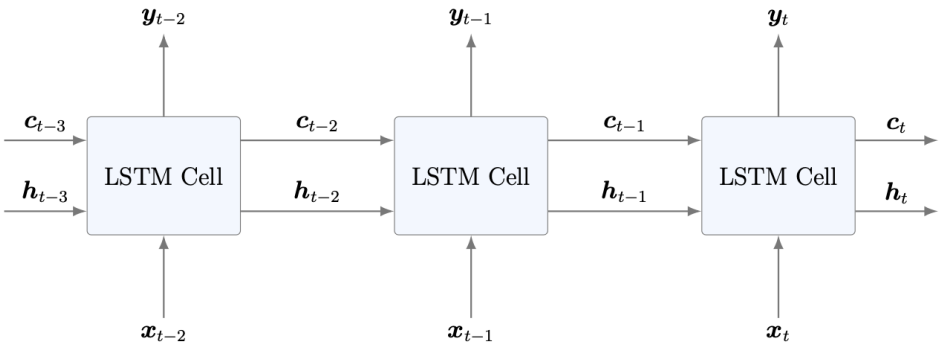

Figure 2: Sequence of LSTM cells

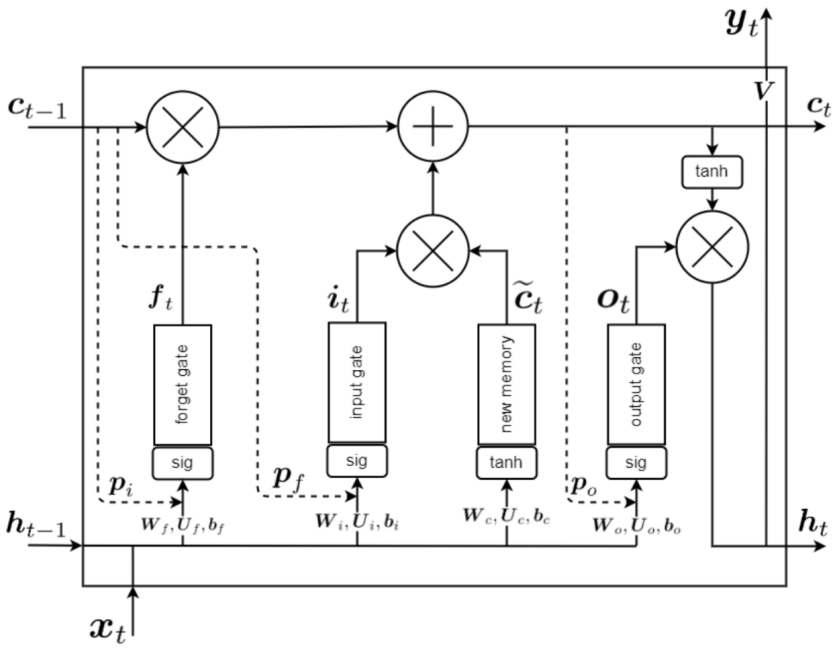

Figure 3: Internal structure of an LSTM cell [4]

At each time step, the LSTM cell computes three gates: the input gate, the forget gate, and the output gate, see Fig. 3. The gates are defined as:

$$i_t = \sigma\left(W_i h_{t-1} + U_i x_t + p_i \odot c_{t-1} + b_i\right)$$

$$f_t = \sigma\left(W_f h_{t-1} + U_f x_t + p_f \odot c_{t-1} + b_f\right)$$

$$o_t = \sigma\left(W_o h_{t-1} + U_o x_t + p_o \odot c_t + b_o\right)$$

where $W_* \in \mathbb{R}^{p \times p}$, $U_* \in \mathbb{R}^{p \times d}$, and $b_* \in \mathbb{R}^p$ are learnable weight matrices and bias vectors, $\sigma(\cdot)$ denotes the sigmoid function applied element-wise, $c_{t-1} \in \mathbb{R}^p$ is the final memory of the last time step. In each gate, a sigmoid function is applied, producing values in $[0, 1]$, allowing them to control how much information is passed or blocked. The vectors $p_* \in \mathbb{R}^p$, are known as *peephole connections*. [4] They allow each gate to directly access the last or the current final memory, thereby enabling the gates to make decisions based not only on the input and hidden state. The symbol $\odot$ denotes the Hadamard (element-wise) product. The equation below defines the gate that combines the current input with the previous hidden state to capture new information:

$$\tilde{c}_t = \tanh\left(W_c h_{t-1} + U_c x_t + b_c\right).$$

Now we can calculate the final memory:

$$c_t = (f_t \odot c_{t-1}) + (i_t \odot \tilde{c}_t)$$

which combines the previous final memory with new information. The forget gate f_t determines how much of the previous final memory c_{t-1} is preserved, while the input gate $i_t \in [0,1]^p$ and $\tilde{c}_t \in [-1,1]^p$ control how much of the new information is added to the final memory. Finally, the hidden state is computed as

$$h_t = o_t \odot \tanh(c_t).$$

3 Data Preparation and Exploration

In this section, we will anlayse the alarm time series dataset [6], perform necessary preprocessing steps, and conduct exploratory data analysis to understand the underlying patterns and characteristics of the data.

3.1 Alarm Dataset

The dataset contains structured alarm event records from monitored industrial processes and assets. Each row represents a single alarm event and includes temporal and descriptive attributes. In total, there are 102,319 alarms. The records span 2015 through 2024 and include the following columns:

Timestamp: Exact date and time at which the alarm event occurred.
ProcessID: Unique identifier of the process that generated the alarm.
AssetID: Identifier of the asset associated with the alarm event.
AlarmSeverityName: Severity level of the alarm, indicating its criticality (High, Medium, Low).
State: Alarm state transition describing the change in condition (Normal to Abnormal, Abnormal to Normal, or Abnormal to Abnormal).
TransactionMessage: Original system-generated message describing the alarm condition in detail.
Stage: Operational lifecycle stage of the alarm, such as Cancelled or Cleared.
AlarmClassName: Classification or category of the alarm reflecting its functional grouping.
Year: Calendar year in which the alarm event occurred.
Month: Calendar month in which the alarm event occurred.
Day: Day of the month on which the alarm event occurred.
DayOfWeek: Day of the week corresponding to the alarm event.
Season: Season during which the alarm event occurred, derived from the event timestamp.
Hour: Hour of the day at which the alarm was triggered.
ProcessedMessage: Cleaned and standardized version of the alarm message, designed to support efficient analysis and downstream processing.

Table 1 summarizes missing values by column. All missing entries occur in string or categorical columns, so we impute them with *Unknown*.

Table 1: Missing Values in the Dataset

	Column	Missing	Missing in %
0	Stage	2984	2.92
1	TransactionMessage	594	0.58
2	ProcessedMessage	594	0.58
3	AssetID	271	0.26

3.2 Feature Analysis

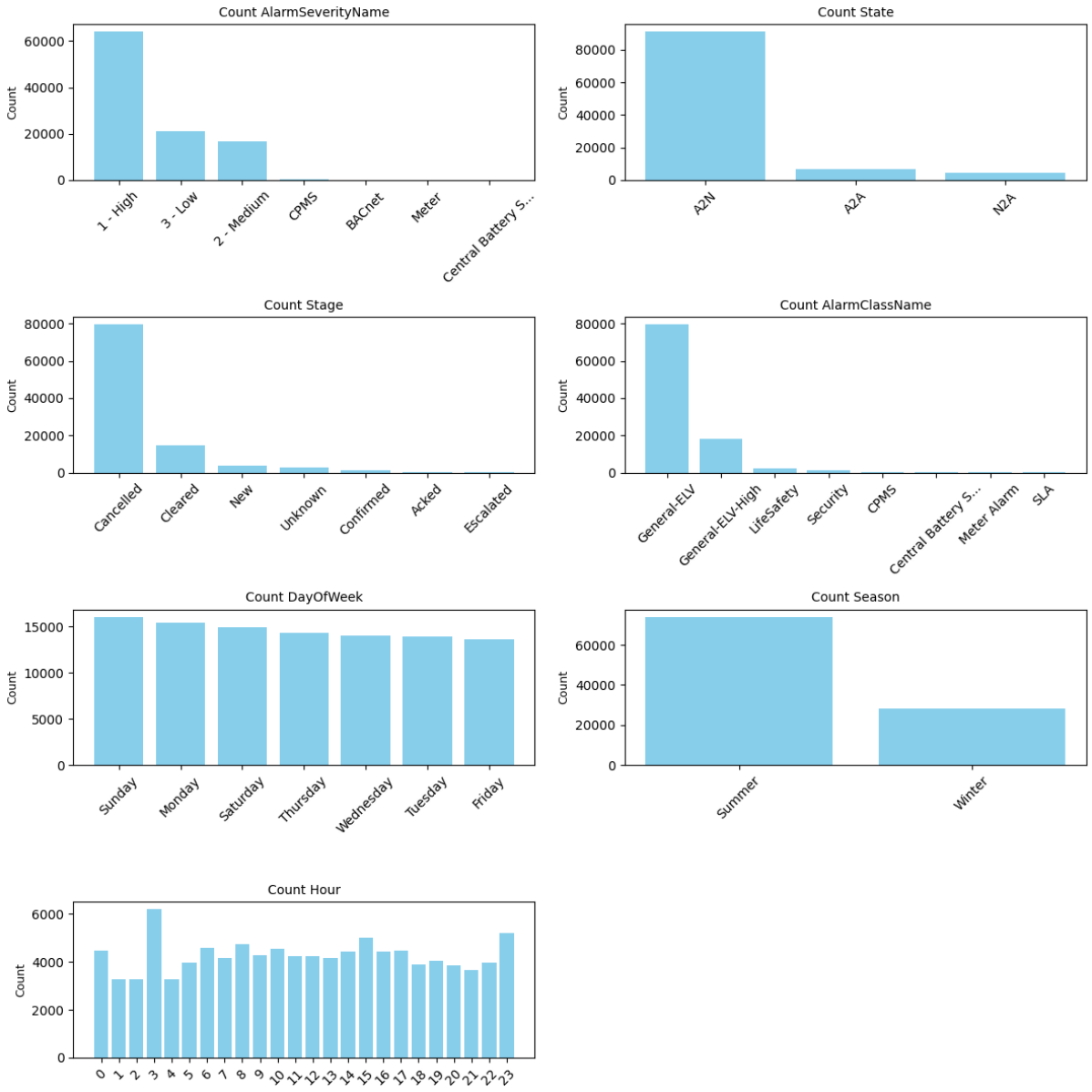

Figure 4: Count Plots for Categorical Variables

Overall, Fig. 4 shows that *High* severity alarms are the most frequent. Most alarms are in the *Abnormal to Normal (A2N)* state and are *Cancelled*. The categorical variables (*AlarmSeverityName, State, Stage, AlarmClassName*) are strongly imbalanced, while temporal features such as

day of week and hour are more evenly distributed. The number of alarms in summer is higher than in winter, indicating a possible seasonal effect.

Table 2: Top 10 AssetIDs

	AssetID	Count
0	AAA-BMS-SSIF	49137
1	1-JK1-JK1-00-D.01-AC-ACON-VAVU-0021	2760
2	1-JK1-JK1-00-D.23-AC-ACON-VAVU-0020	1944
3	1-JK1-JK1-00-C.27-AC-ACON-VAVU-0019	1879
4	1-JK1-JK1-00-E.03-AC-ACON-VAVU-0024	1805
5	1-JK1-JK1-00-E.03-AC-ACON-VAVU-0023	1779
6	1-JK1-JK1-00-D.01-AC-ACON-VAVU-0022	1578
7	1-JK1-JK1-00-D.17-AC-ACON-VAVU-0017	1456
8	0-JK1-JK1-B2-2.04-AC-ACON-AHU-0009	1455
9	1-JK1-JK1-00-D.04-AC-ACON-VAVU-0012	1371

Table 3: Top 10 Processed Messages

	ProcessedMessage	Count
0	device offline	53061
1	temperature set point	7301
2	high temperature alarm	4514
3	fan command alarm	2769
4	vavj0900029 space temp alarm	2750
5	temperature high	2699
6	vavj0900028 space temp alarm	1939
7	vavj0900027 space temp alarm	1872
8	vavj0900033 space temp alarm	1800
9	vavj0900032 space temp alarm	1767

Table 2 lists the 10 most frequent AssetIDs. The most common is "AAA-BMS-SSIF" with 49,137 occurrences. Without further information, it's difficult to explain why this asset generates so many alarms. Further inspection of asset types and associated processes is needed to get a better understanding of the data.

Table 3 lists the 10 most frequent processed alarm messages. The most common is *device offline* (53,061 occurrences). While this indicates connectivity or availability issues, the dataset lacks context about the specific device or root causes. Other frequent messages are temperature-related alarms.

3.3 Daily Alarm

The alarms are aggregated on a daily basis. Fig. 5 illustrates the daily number of alarms from 2015 to 2024. Alarm activity is low in the early years, shows increasing variability from 2018 onward, and exhibits a clear rise in both frequency and magnitude after 2021, with several spikes during 2023–2024.
Fig. 6 presents daily alarm counts exceeding 200. Only a small number of days surpass this threshold, with one day showing an exceptionally high count of more than 1,000 alarms.

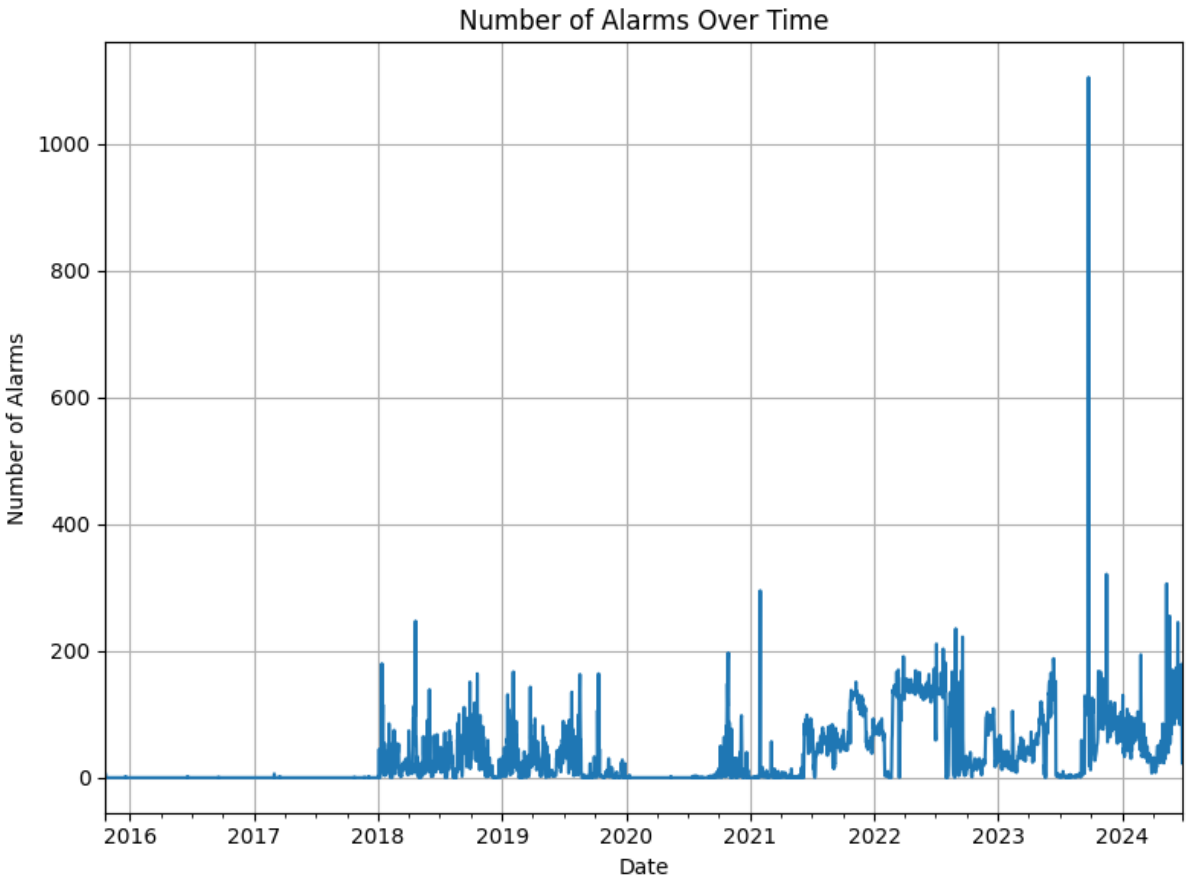

Figure 5: Number of Alarms Over Time

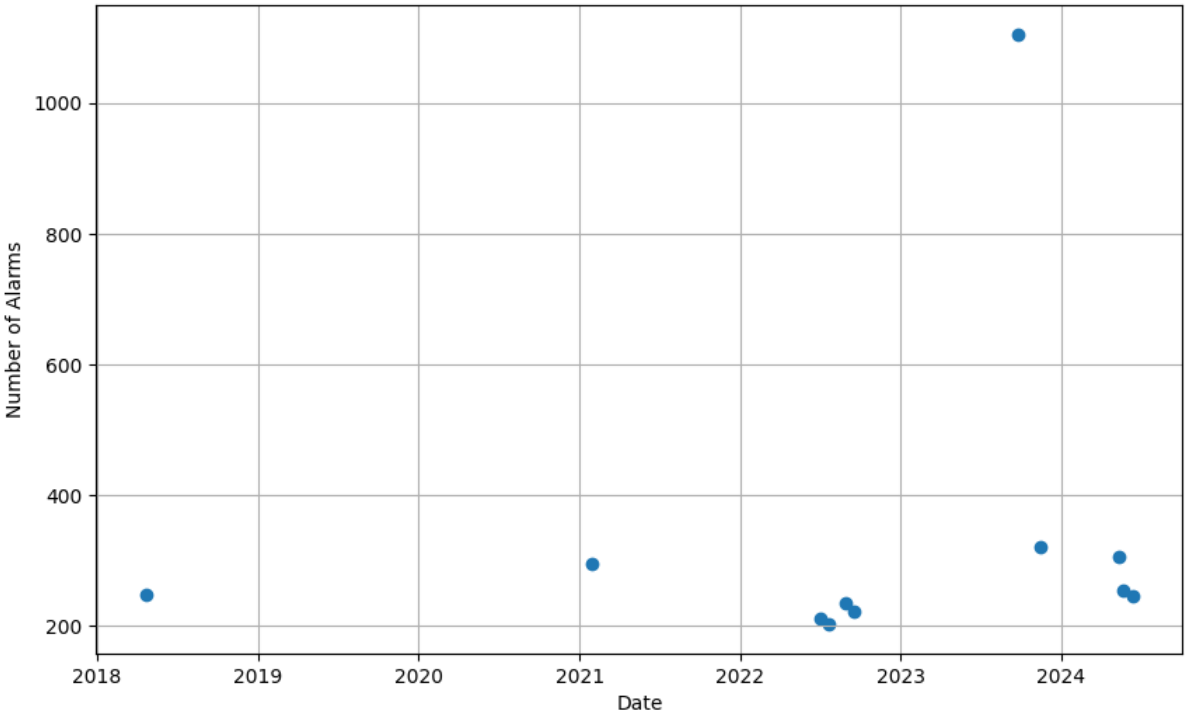

Figure 6: Number of Alarms exceeding 200 per Day

3.4 Seasonal-Trend decomposition

STL (Seasonal-Trend decomposition using LOESS) [7], which separates the series into a smooth long-term trend, a repeating seasonal component, and a residual. In this case, the trend indicates a steady increase in daily alarm counts over time, while the seasonal component shows no strong recurring pattern. Fig. 4 shows higher alarm counts in summer than in winter, but this seasonal difference does not appear as a strong periodic signal in the STL plot. The residuals capture irregular fluctuations beyond trend and seasonality, and their variability appears to grow over time.

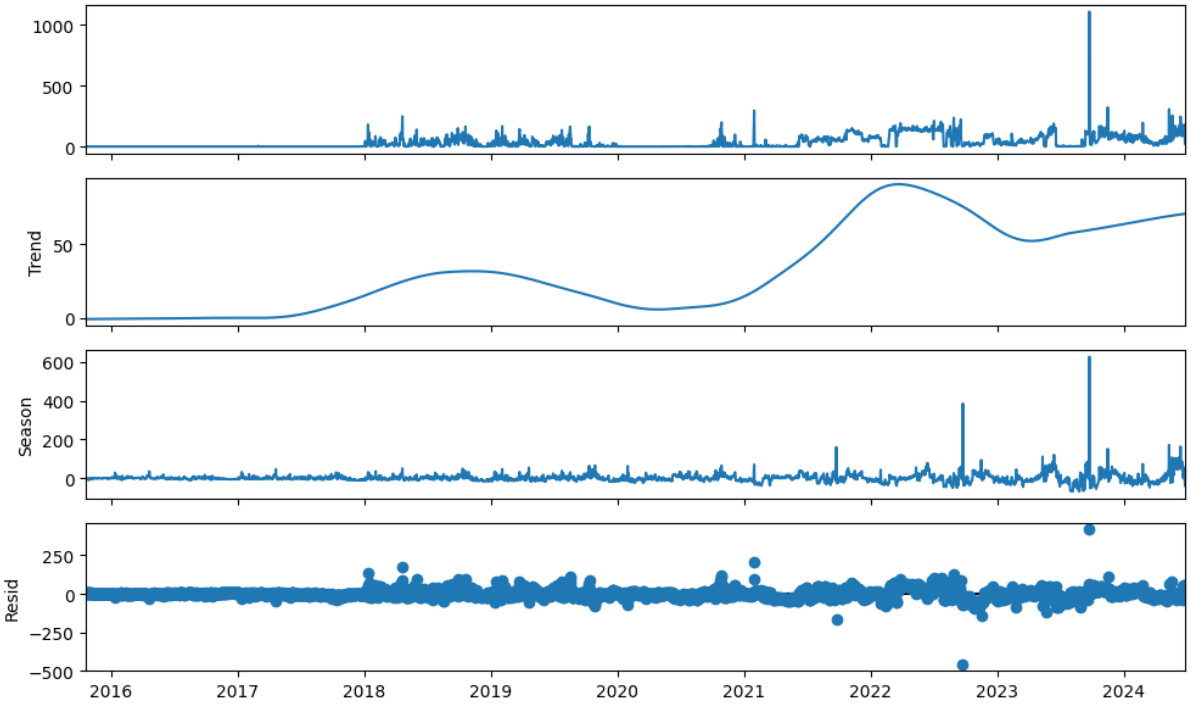

Figure 7: STL Decomposition of Daily Alarms

3.5 Stationarity

In time series analysis, it is important for the time series to be stationary. The KPSS test is a statistical test for stationarity.

- Null hypothesis: the series is stationary (trend-stationary).
- Alternative: the series is non-stationary.

So a small p-value suggests non-stationarity, while a large p-value means you fail to reject stationarity.

The KPSS test of the *daily_alarms* indicates that the differenced series is stationary. The p-value is 0.1, which is above the 0.05 significance threshold, so we fail to reject the null hypothesis of stationarity and proceed under the assumption that the differenced series is stationary.

3.6 ACF and PACF

Fig. 8 presents the ACF and PACF of the differenced daily alarm time series. The ACF shows significant correlations at the first two lags, while the PACF is significant only at the initial lags

and then decays, which is characteristic for a MA-process. Overall, this pattern suggests only short-term dependencies in the data.

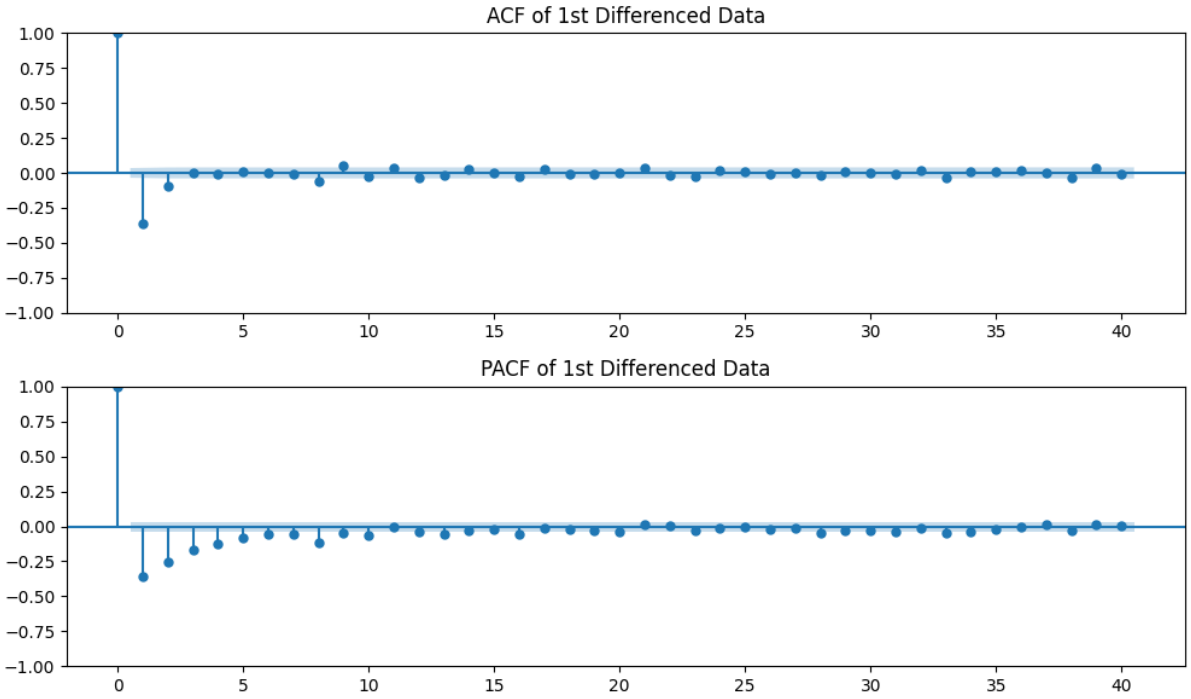

Figure 8: ACF and PACF of Differenced Daily Alarms

3.7 Power Spectral Density

Fig. 9 illustrates the power spectral density of the differenced daily alarm time series across multiple zoom levels of the x-axis. The spectrum is shown on a period scale (the inverse of frequency), so a 7-day period corresponds to a weekly cycle. The plot does not show clear *weekly*, *monthly*, or *yearly* cycles. The plot concentrates more power at shorter periods, but it does not show clear peaks that would indicate strong periodic behavior.

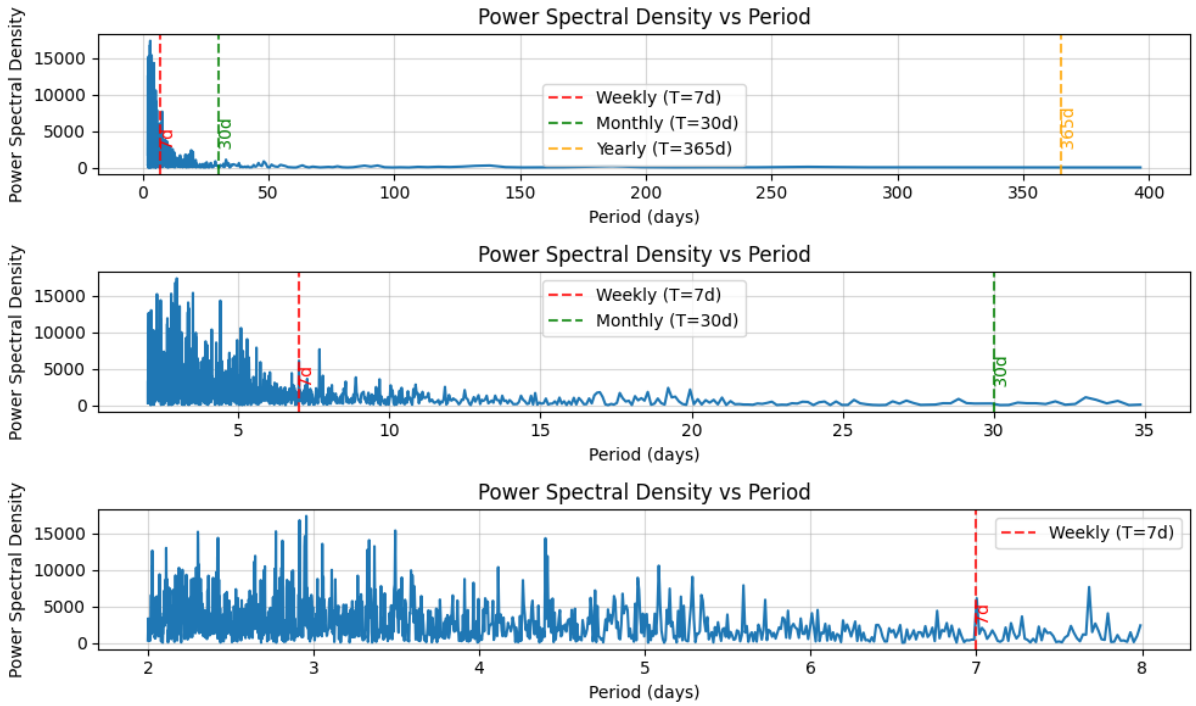

Figure 9: Power spectral density of the differenced daily alarm time series at multiple zoom levels of the period axis.

3.8 Grouping

Although the dataset includes an *AlarmClassName* variable that groups alarms, it is highly imbalanced. To obtain a more informative categorical representation, we introduce a new variable, *AlarmGroup*, derived from ProcessedMessage. The *ProcessedMessage* column contains 86 unique values. To use this information as a categorical feature, alarms are grouped based on semantic meaning and severity, resulting in 14 distinct alarm groups. This grouping was performed manually rather than via automated clustering. Meaningful alarm categorization requires domain knowledge.

Fig. 10 shows the distribution of alarms across the defined groups. The most frequent group is *device offline* (53,061 occurrences), followed by *temperature*-related alarms. *Gas* alarms occur less frequently, but they may be safety-critical. The *trouble* group occurs 1,246 times; however, its exact operational meaning cannot be determined with certainty from the available information.

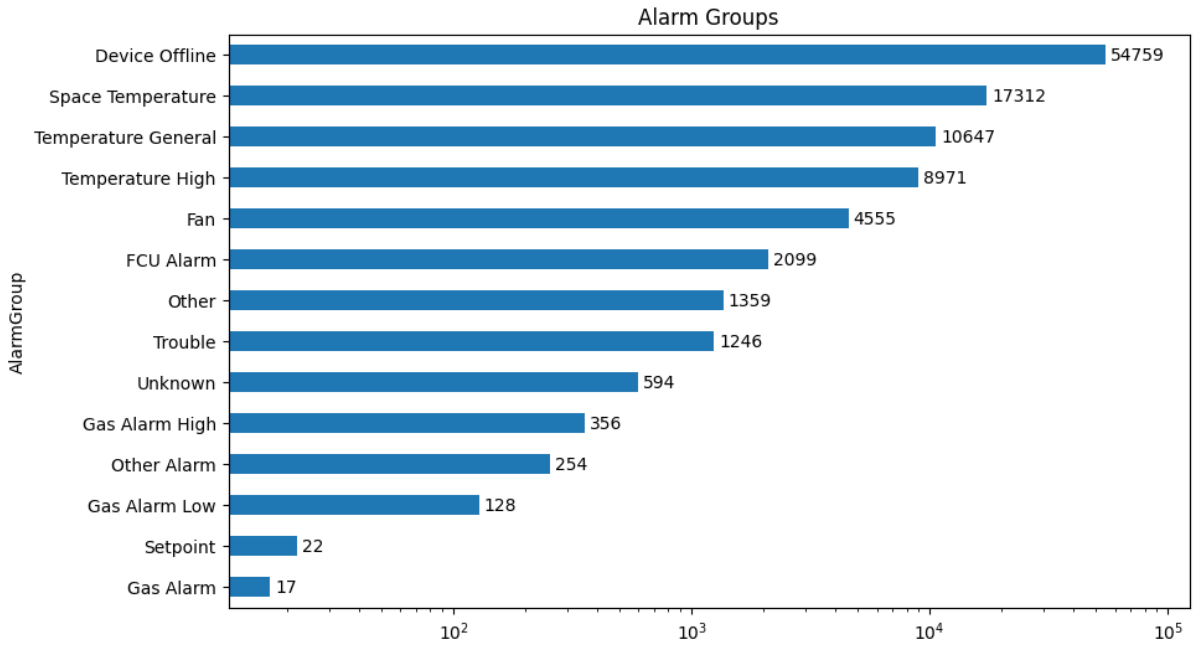

Figure 10: Alarm Groups

The dataset contains 1,604 unique *AssetID* values, which is too many for a categorical variable. We considered grouping by keeping the top 10 assets and combining the remainder into *other*. Fig. 11 shows the asset counts. Even after grouping, *AssetID* remains highly imbalanced, so to keep the feature set small we drop this variable.

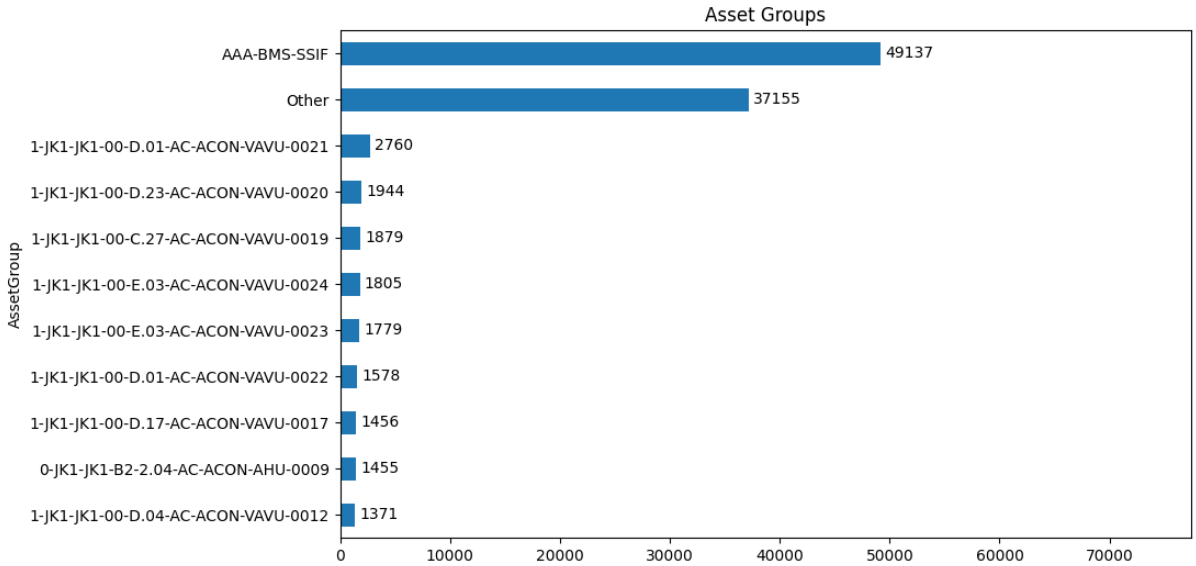

Figure 11: Asset Groups

3.9 One-Hot Encoding

Categorical features are one-hot encoded, including *AlarmSeverityName, State, Stage, AlarmClassName, DayOfWeek, Season*, and *AlarmGroup*. In addition, calendar-based time features (*Year, Month, and Day*) are added as numerical variables. Table 4 summarizes the number of

features per category. After one-hot encoding, the exogenous feature set contains 52 variables. We apply a one-step lag to exogenous variables, i.e., we predict the next day's alarm count using the previous time step's exogenous inputs. Time-derived calendar features such as *Year, Month, Day, DayOfWeek*, and *Season* are not lagged.

Table 4: Number of Exogenous Features by Category

	Category	Count
0	AlarmSeverityName	7
1	State	3
2	Stage	7
3	AlarmClassName	9
4	DayOfWeek	7
5	Season	2
6	AlarmGroup	14
7	Year	1
8	Month	1
9	Day	1

4 Results

We perform time series forecasting using a supervised learning framework in which future values are predicted based on historical observations. The raw data are first aggregated to a daily frequency. The forecasting horizon is one day ahead, meaning the model predicts the value at day $(t+1)$ using information available up to day (t).

Model performance is assessed with time series cross-validation, which respects temporal order. The data are split into multiple sequential folds; each fold trains on earlier observations and evaluates on a later 365-day hold-out window. RMSE is used as the evaluation metric to penalize larger prediction errors more strongly, which is particularly important for accurately capturing sharp spikes in the time series.

4.1 Baseline Model

As a baseline, we use a model that predicts the alarm count of the next day using the observed value from the previous day. Formally, let y_t denote the alarm count at day t. The baseline forecast is defined as

$$\hat{y}_{t+1} = y_t.$$

Using time-series cross-validation, we compute the RMSE for each fold as well as the average RMSE across all folds.

```
RMSE per Fold:
Fold 1: 19.14
Fold 2: 26.47
Fold 3: 16.59
Fold 4: 28.31
Fold 5: 81.82
Average RMSE: 34.47
```

4.2 SARIMAX

Based on the KPSS test results reported above, first-order differencing is sufficient to achieve stationarity; therefore we set $d = 1$.

A common method for selecting the AR and MA orders is to inspect the autocorrelation function ACF and partial autocorrelation function PACF plots. Fig. 8 shows a sharp cutoff after two lags in the ACF and a decay in the PACF, indicating the differenced time series follows an MA(2) process.

A plausible non-seasonal configuration is $(p, d, q) = (0, 1, 2)$. As no clear seasonal pattern is observed, the seasonal component is omitted by setting $(P, D, Q) = (0, 0, 0)$.

4.2.1 Alarms

We first fit a SARIMAX model without exogenous variables. Fig. 12 presents the time series cross-validation results for this model. For each fold, the predicted values are plotted against the corresponding observed values. For clarity, we display only the test folds (one year per fold) rather than the full time-series history, which allows a more detailed comparison between predicted and actual values. The same procedure is applied to all other models.

```
RMSE per Fold:
Fold 1: 17.41
Fold 2: 24.03
Fold 3: 18.82
Fold 4: 29.46
Fold 5: 70.92
Average RMSE: 32.13
```

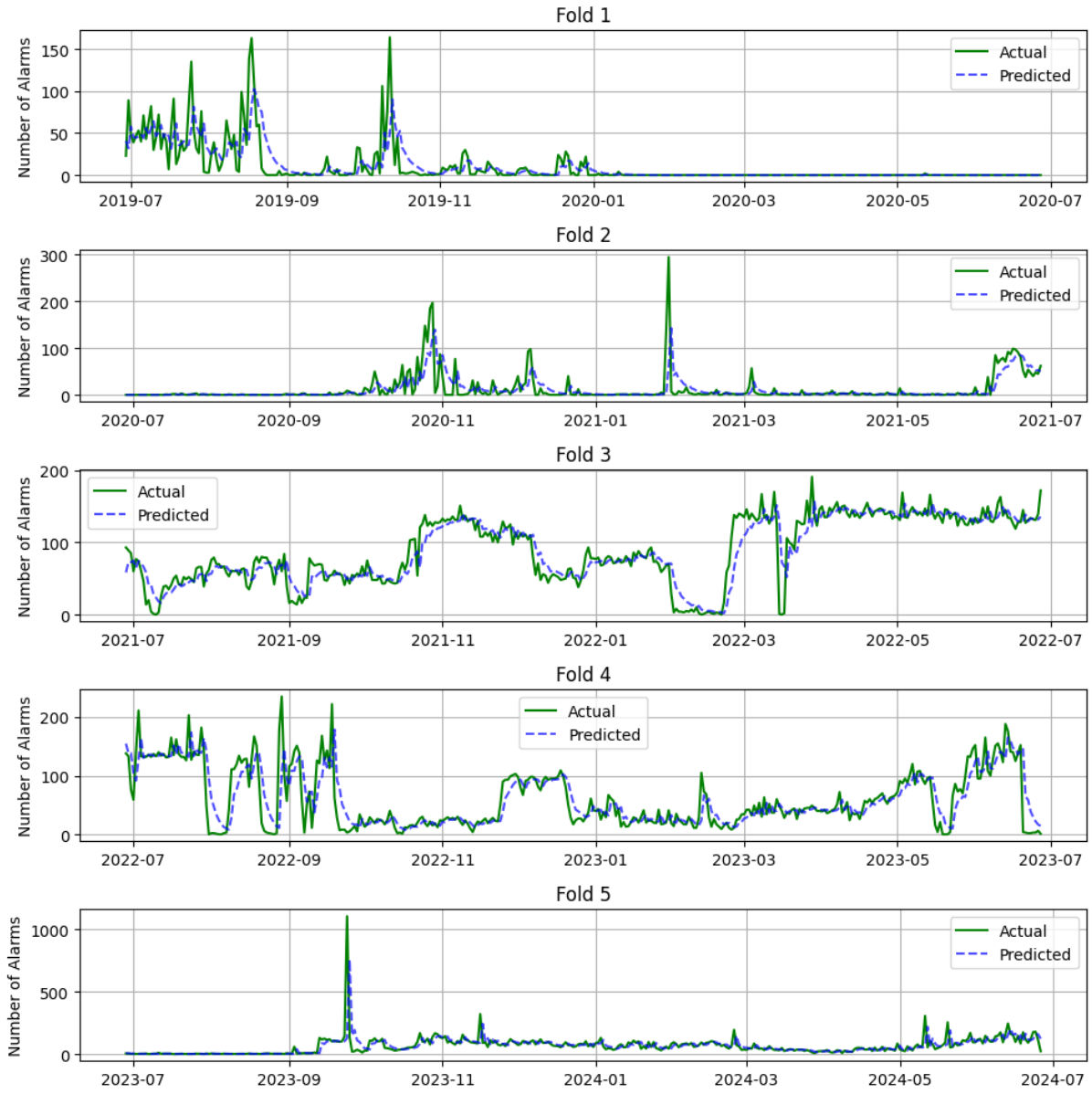

Figure 12: SARIMAX Actual vs Predicted for All Folds - only Daily Alarms

4.2.2 Alarms with Exogenous Variables

Next, we fit a SARIMAX model with all exogenous variables from Table 4. Fig. 13 shows the time series cross-validation results.

```
RMSE per Fold:
Fold 1: 17.92
Fold 2: 49.96
Fold 3: 51.15
Fold 4: 30.04
Fold 5: 93.68
Average RMSE: 48.55
```

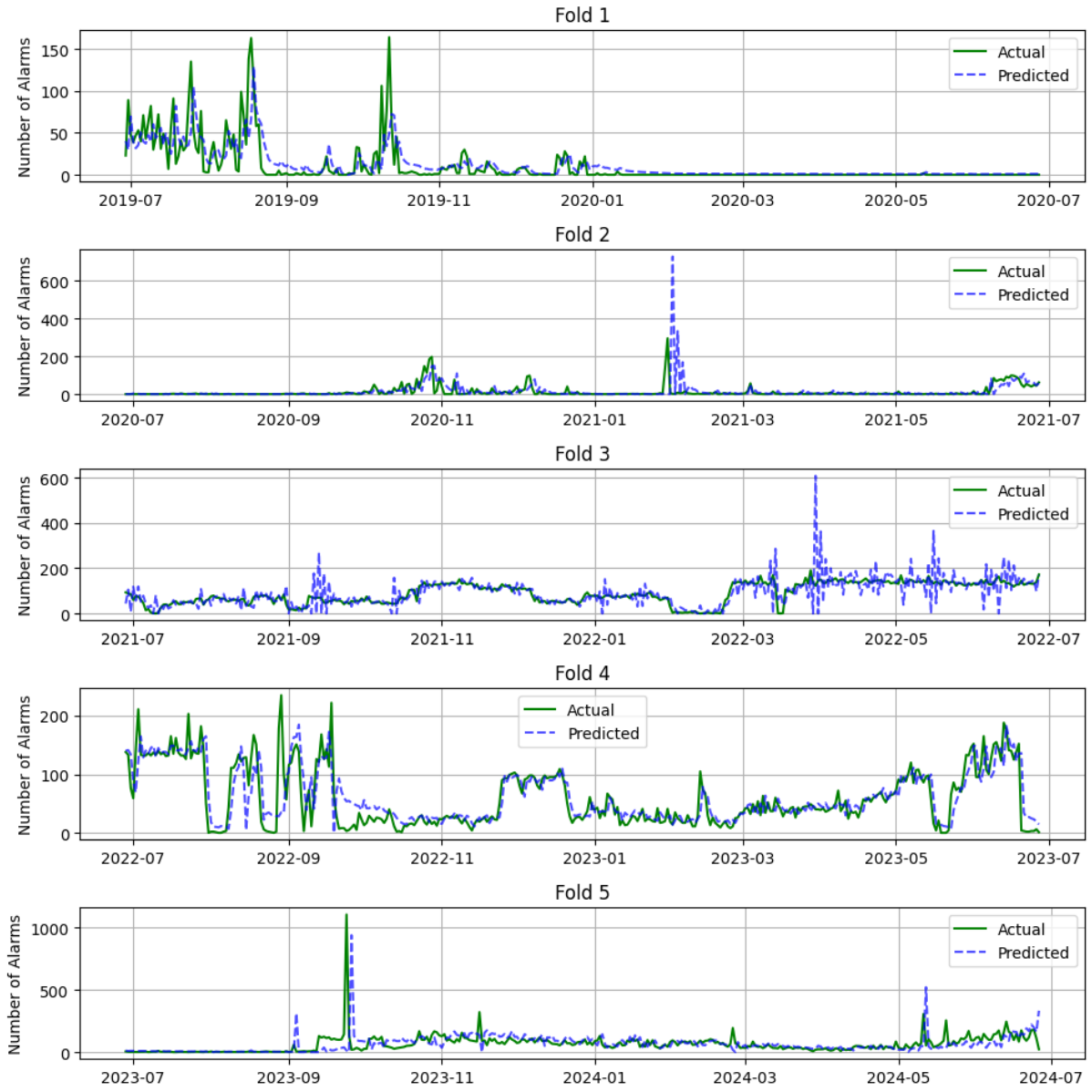

Figure 13: SARIMAX Actual vs Predicted for All Folds with Exogenous Variables

4.2.3 Alarms with selected Exogenous Variables

To reduce dimensionality, we select only the most relevant exogenous variables using forward selection based on the Akaike Information Criterion (AIC). Model performance is evaluated using time series cross-validation, where the first four folds are used for validation during model selection and the final fold is used as a test set. The selected variables are *AlarmSeverityName*, *State*, *DayOfWeek*, and *AlarmClassName*. This reduced set comprises 26 features, compared to the original 52. Fig. 14 shows the time-series cross-validation results of the model using the selected exogenous variables.

```
RMSE per Fold:
Fold 1: 17.43
Fold 2: 24.52
Fold 3: 19.61
Fold 4: 30.25
```

Fold 5: 68.40
Average RMSE: 32.04

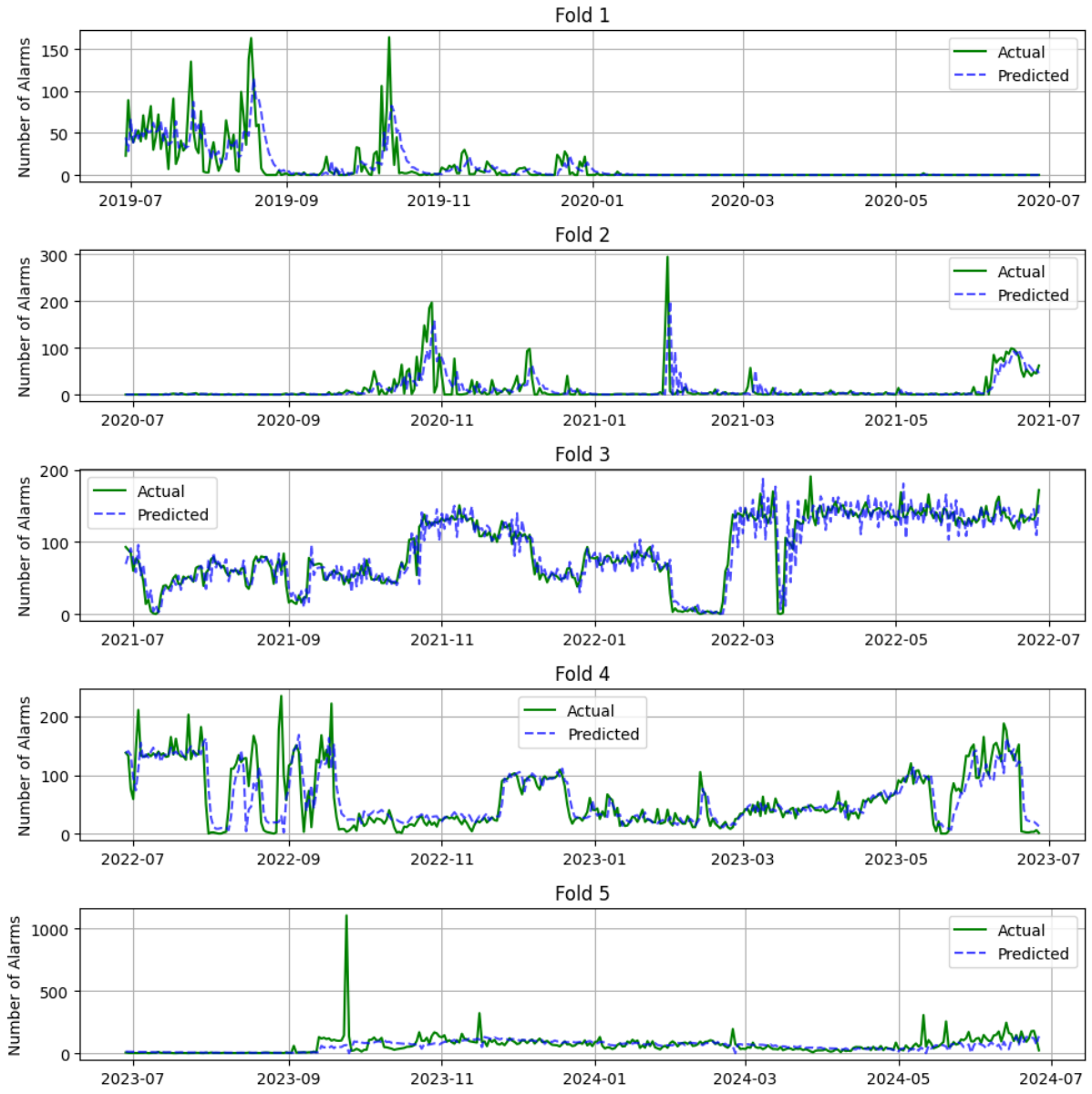

Figure 14: SARIMAX Actual vs Predicted for All Folds - Selected Exogenous Features

Conclusion

The SARIMAX model without exogenous variables performs slightly better than the baseline model. However, adding all exogenous variables worsens the results because it introduces additional noise. Using only the selected variables gives the best overall performance. Nevertheless, the model still lags behind the true values, especially when sharp spikes occur.

4.3 XGBoost

For this model, additional lagged features of the daily alarm series are included as input variables, covering lags from one to seven days. This means that the model can learn from the past alarm counts to make predictions for the current day.

Fig. 15 shows the time series cross-validation results of the XGB Regressor model with exogenous variables and lagged features.

```
RMSE per Fold:
Fold 1: 18.67
Fold 2: 25.70
Fold 3: 29.40
Fold 4: 29.04
Fold 5: 74.68
Average RMSE: 35.50
```

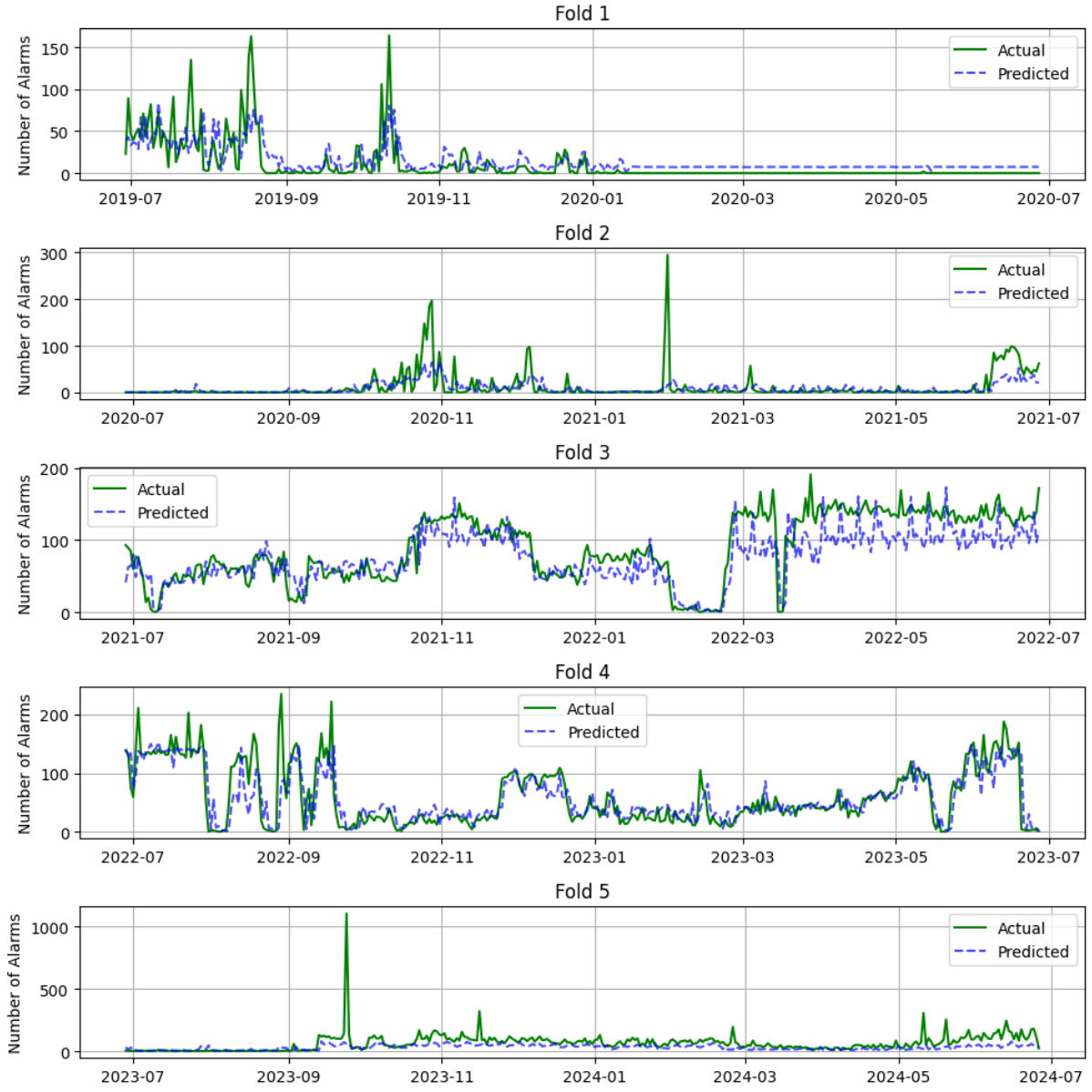

Figure 15: XGB Regressor Actual vs Predicted for All Folds

The XGBoost model underperforms compared to the SARIMAX models; however, it is still able to capture the general trend and variability in the alarm counts.

The permutation feature importance indicates how much each feature contributes to the model's predictive performance. Fig. 16 shows the feature importance of the Top 20 XGB Regressor model. The most important features are Stage_Cancelled, AlarmClassName_General-ELV, followed by lagged alarm count features.

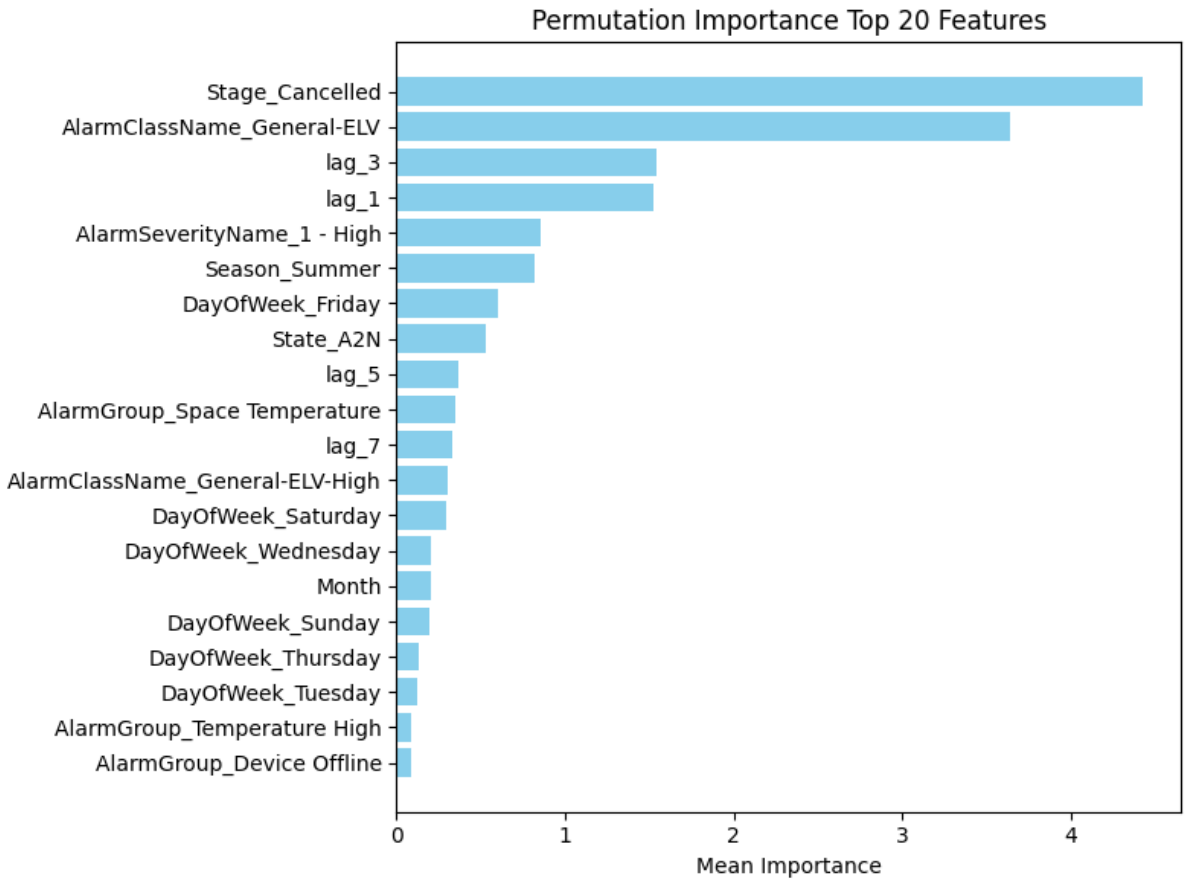

Figure 16: XGB Regressor Feature Importance

4.4 RNN

For the RNN, we use a fixed lookback window of the past days of all variables to predict the alarm count of the next day. The model consists of a recurrent layer with a ReLU activation, followed by dropout and a linear output layer. The hyperparameters include the *lookback window*, *batch size*, and *learning rate*, as well as recurrent-model parameters such as *hidden size* and the *number of layers*. Models are trained for up to 200 epochs with early stopping based on validation performance. Model performance is evaluated using time series cross-validation, where the first four folds are used for validation during model selection and the final fold is used as a test set. The hyperparameters are tuned over the following ranges:

- Lookback window: 1–20
- Batch size: $16, 32, 64, 128$
- Hidden size: 1–32
- Number of layers: 1–3
- Learning rate: 0.01–0.0001

4.4.1 Alarms

We first fit a RNN model without exogenous variables. Fig. 17 presents the time series cross-validation results for this model.

```
RMSE per Fold:
Fold 1: 17.50
Fold 2: 22.89
Fold 3: 18.32
Fold 4: 27.22
Fold 5: 67.56
Average RMSE: 30.70
```

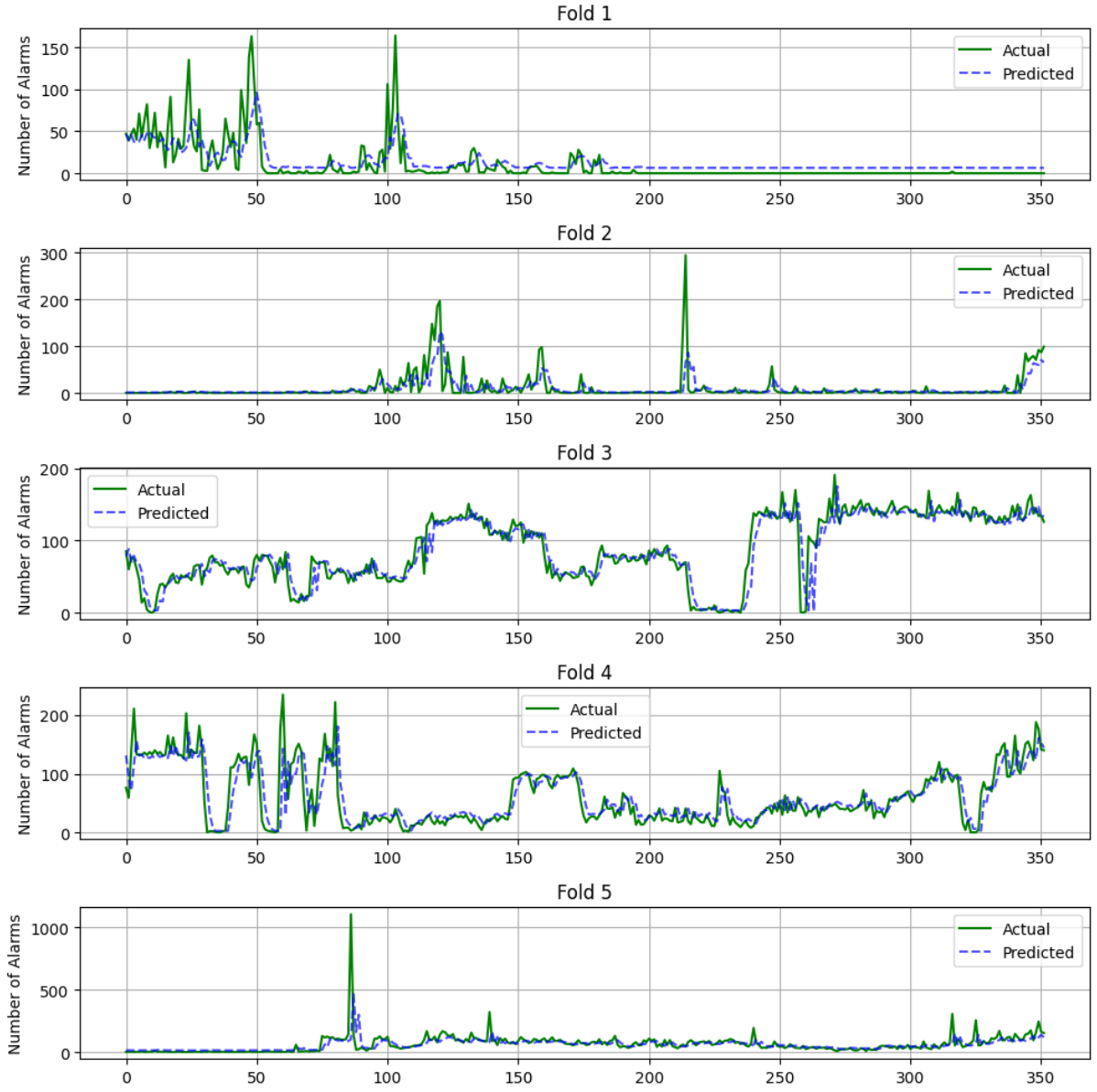

Figure 17: RNN Actual vs Predicted for All Folds - only Daily Alarms

The optimized hyperparameters for the RNN model are:

- Lookback window: 3
- Batch size: 16
- Hidden size: 18
- Number of layers: 3
- Learning rate: 9.9×10^{-3}

4.4.2 Alarms with Exogenous Variables

We fit a RNN model with all exogenous variables Fig. 18 shows the time series cross-validation results.

```
RMSE per Fold:
Fold 1: 19.69
Fold 2: 27.35
Fold 3: 42.30
Fold 4: 48.98
Fold 5: 84.09
Average RMSE: 44.48
```

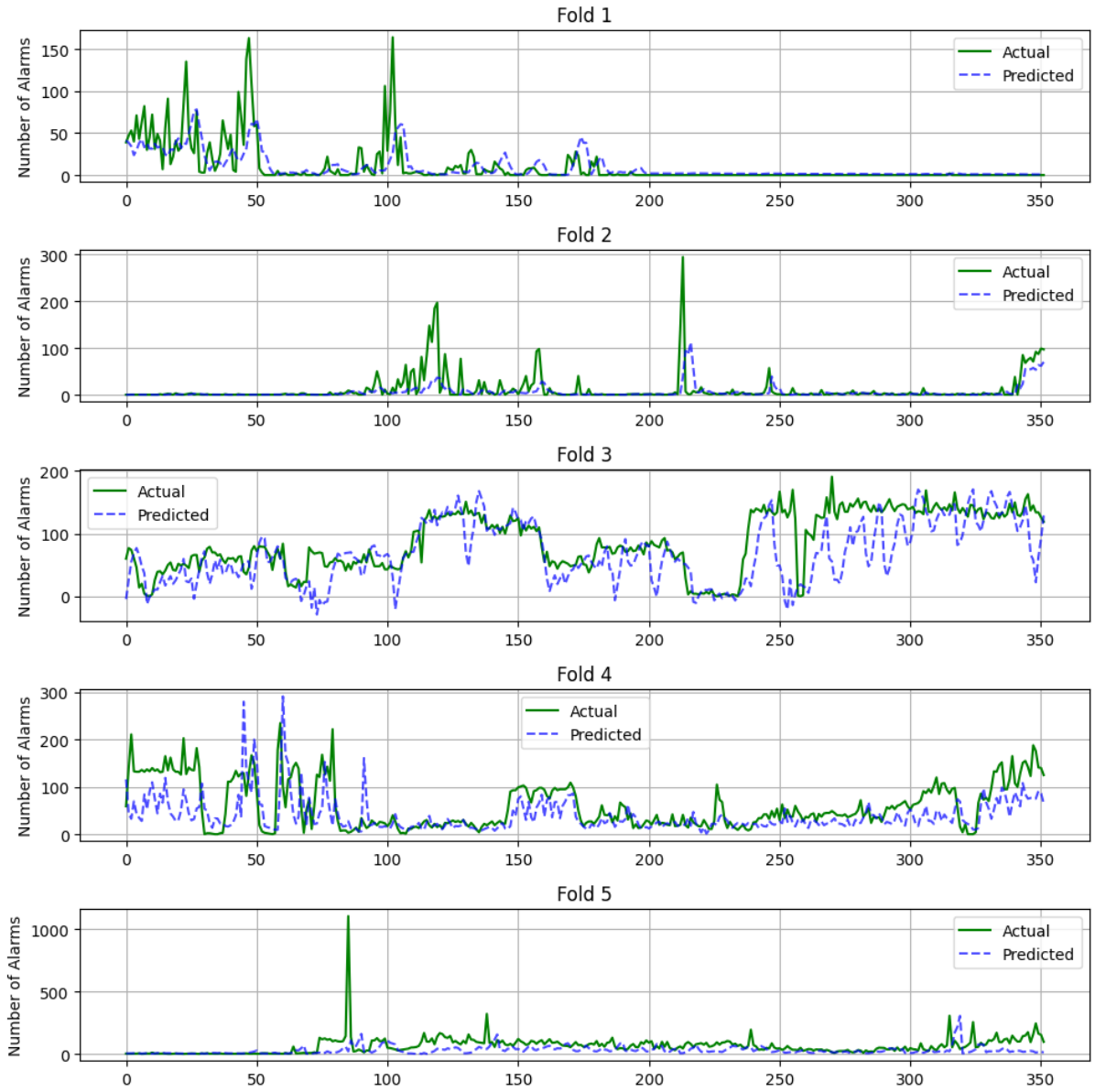

Figure 18: RNN Actual vs Predicted for All Folds with Exogenous Variables

The optimized hyperparameters for the RNN model are:

- Lookback window: 4
- Batch size: 16
- Hidden size: 5
- Number of layers: 1
- Learning rate: 6.7×10^{-4}

Conclusion

As observed for the SARIMAX models, including too many exogenous variables worsen performance by introducing additional noise. The RNN model without exogenous variables performs better than the variants that includes them. The lookback window of this model is 3 days, which

means recent observations contain most of the relevant information for predicting the next day's alarm count.

Both RNN-based models are optimized using Optuna with 50 trials and trained on an NVIDIA A100 GPU. The total optimization runtime is approximately 33 minutes and 42 minutes, respectively.

4.5 LSTM

For the LSTM model, we use the same data preprocessing, training, hyperparameter optimization, and evaluation setup as for the RNN, replacing the recurrent with an LSTM layer. Another change is that the lookback window is increased to 128 days in order to explore potential long-term dependencies.

4.5.1 Alarms

Fig. 19 presents the time series cross-validation results for this model.

```
RMSE per Fold:
Fold 1: 17.25
Fold 2: 23.58
Fold 3: 45.02
Fold 4: 26.31
Fold 5: 61.07
Average RMSE: 34.65
```

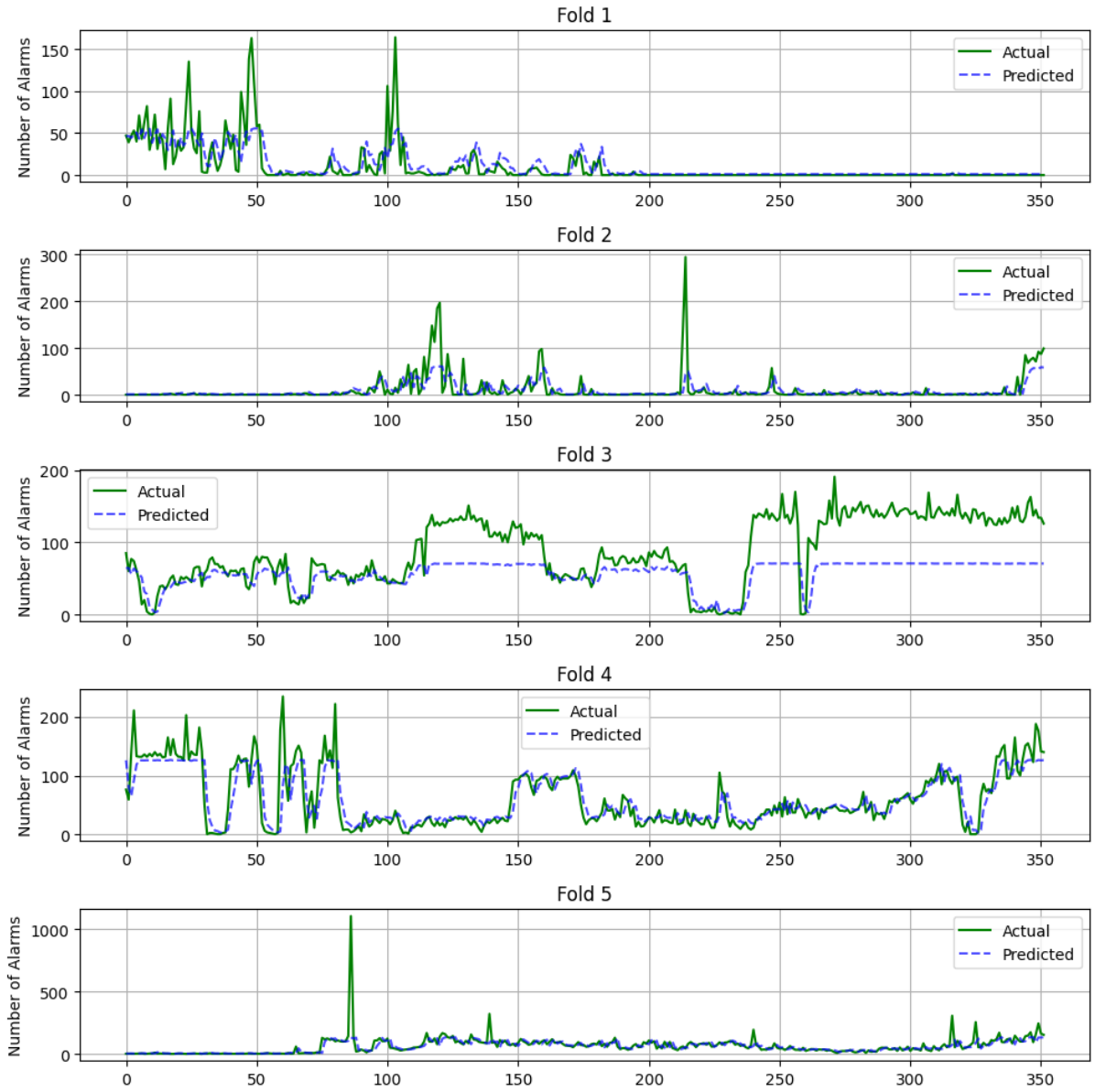

Figure 19: LSTM Actual vs Predicted for All Folds

The optimized hyperparameters for the LSTM model are:

- Lookback window: 7
- Batch size: 16
- Hidden size: 25
- Number of layers: 1
- Learning rate: 3.4×10^{-3}

4.5.2 Alarms with Exogenous Variables

Fig. 20 shows the time series cross-validation results of the model with exogenous variables.

```
RMSE per Fold:
Fold 1: 23.88
Fold 2: 27.59
```

Fold 3: 52.42
Fold 4: 46.02
Fold 5: 74.72
Average RMSE: 44.93

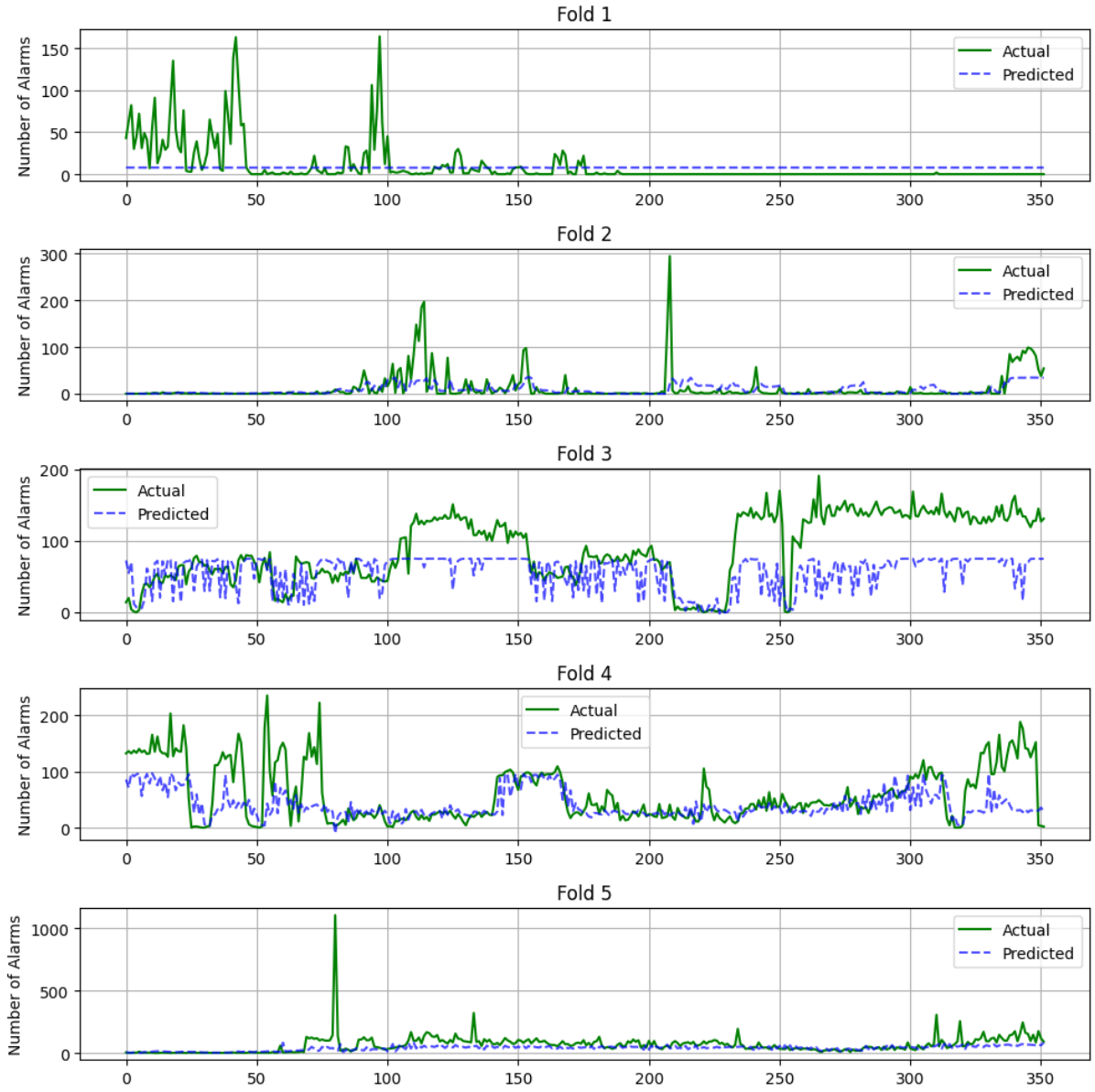

Figure 20: LSTM Actual vs Predicted for All Folds

The optimized hyperparameters for the LSTM model are:

- Lookback window: 9
- Batch size: 16
- Hidden size: 106
- Number of layers: 2
- Learning rate: 9.9×10^{-4}

Conclusion

Even though the lookback window search range is larger, the model consistently prefers smaller lookback values of 7 and 9. Once again, the model without exogenous variables achieves better performance. Overall, the LSTM performs worse than the RNN on the validation set, but shows improved performance on the test set (last fold).

Hyperparameter optimization is carried out using Optuna with 50 trials, resulting in total runtimes of approximately 57 minutes and 70 minutes, respectively, on an NVIDIA A100 GPU.

5 Conclusion

In this project, we evaluated several time-series forecasting models to predict daily alarm counts based on historical patterns and exogenous variables. To reduce computational complexity, the data were aggregated at a daily level, and the models were trained to predict the alarm count for the following day. Visualizing the data helps to understand the overall pattern, trends, and unusual events. The total runtime of the complete pipeline was approximately 5 hours. Time series cross-validation is computationally expensive, feature selection, and neural network models in particular require more training time than classical approaches, especially when hyperparameter optimization is performed.

The data shows non-stationary behavior with an upward trend but lacked clear seasonal patterns. First-order differencing was sufficient to achieve stationarity. Analysis of the ACF and PACF, along with power spectral density plots, indicates short-term dependencies, with only the most recent days contributing meaningful predictive information. Categorical features were included using one-hot encoding. We evaluate all models using time-series cross-validation with 5 folds. For models that involve model selection, the first four folds are used for validation during model selection, and the final fold is used as a test set. This includes SARIMAX (selected exog), RNN, and LSTM:

The results for each model by fold, along with the average RMSE, are summarized in Table 5. It shows that models using only the alarm history generally perform better than models that include all exogenous variables. Although RNN (alarm) achieves the lowest average RMSE across all folds, this result should be interpreted with caution, since we used the first four folds for validation. Therefore, performance may be slightly optimistic, and the most reliable measure of generalization is the RMSE on Fold 5. LSTM (alarm) performs best on the last fold, although the validation error remains high compared to the other folds. SARIMAX without exogenous variables, and with selected exogenous variables, also have sligthly better results than the baseline model.

Adding all exogenous variables clearly worsens performance. This can be seen for SARIMAX (exog) as well as for RNN (exog) and LSTM (exog), which have much higher RMSE values and less stable results across folds. This suggests that the additional variables introduce noise and do not provide useful extra information in this setting.

Fold 5 is the most difficult for every model, with much higher RMSE values than the earlier folds. This indicates that the last time period is harder to predict, likely because it contains unusual changes or strong spikes that the models cannot learn well from the earlier data.

Across all models, extreme spikes are hard to predict. When a spike is reflected in the forecast, it often appears with a delay, meaning the model reacts after the true increase has already happened. The results also show that the models have problems predicting extreme spikes. In

Table 5: RMSE by model across 5 time-series cross-validation folds (best per column in bold).

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Avg
Baseline	19.14	26.47	**16.59**	28.31	81.82	34.47
SARIMAX (alarm)	17.41	24.03	18.82	29.46	70.92	32.13
SARIMAX (exog)	17.92	49.96	51.15	30.04	93.68	48.55
SARIMAX (selected exog)	17.43	24.52	19.61	30.25	68.40	32.04
XGBoost	18.67	25.70	29.40	29.04	74.68	35.50
RNN (alarm)	17.50	**22.89**	18.32	27.22	67.56	**30.70**
RNN (exog)	19.69	27.35	42.30	48.98	84.09	44.48
LSTM (alarm)	**17.25**	23.58	45.02	**26.31**	**61.07**	34.65
LSTM (exog)	23.88	27.59	52.42	46.02	74.72	44.93

such cases, a more complex model or a special method for rare events may be needed, because simpler forecasting models often smooth sudden changes or react too late. Additional descriptive features can be extracted from each sliding window (e.g., minimum, maximum, peak counts). Exogenous variables should also be added carefully. Irrelevant variables can add noise and reduce generalization, so feature selection is often necessary. Domain knowledge could help identify early signals or leading indicators.

6 References

[1] Granville Tunnicliffe Wilson. "Time Series Analysis: Forecasting and Control, 5th Edition, by George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel and Greta M. Ljung, 2015. Published by John Wiley and Sons Inc., Hoboken, New Jersey, pp. 712. ISBN: 978-1-118-67502-1". In: *Journal of Time Series Analysis* 37.5 (2016), pp. 709–711. DOI: https://doi.org/10.1111/jtsa.12194. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jtsa.12194. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/jtsa.12194.

[2] Fahad Radhi Alharbi and Denes Csala. "A Seasonal Autoregressive Integrated Moving Average with Exogenous Factors (SARIMAX) Forecasting Model-Based Time Series Approach". In: *Inventions* 7.4 (2022). ISSN: 2411-5134. DOI: 10.3390/inventions7040094. URL: https://www.mdpi.com/2411-5134/7/4/94.

[3] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *CoRR* abs/1603.02754 (2016). arXiv: 1603.02754. URL: http://arxiv.org/abs/1603.02754.

[4] Benyamin Ghojogh and Ali Ghodsi. *Recurrent Neural Networks and Long Short-Term Memory Networks: Tutorial and Survey*. 2023. arXiv: 2304.11461 [cs.LG]. URL: https://arxiv.org/abs/2304.11461.

[5] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[6] H. G. Sudhanva. *Industrial Alarm Monitoring Dataset (2018–2024)*. 2024. URL: https://www.kaggle.com/datasets/sudhanvahg/industrial-alarm-monitoring-dataset-2018-2024 (visited on 01/26/2026).

[7] Robert B. Cleveland et al. "STL: A Seasonal-Trend Decomposition Procedure Based on Loess (with Discussion)". In: *Journal of Official Statistics* 6 (1990), pp. 3–73.