Journal of COMPUTER AND FORENSIC SCIENCES

e-ISSN 2956-0799

Volume 4 • Issue 1 • 2025

UNIVERSITY OF CRIMINAL INVESTIGATION AND POLICE STUDIES
Belgrade

UNIVERSITY OF CRIMINAL INVESTIGATION AND POLICE STUDIES, BELGRADE – THE REPUBLIC OF SERBIA
KRIMINALISTIČKO-POLICIJSKI UNIVERZITET, BEOGRAD – REPUBLIKA SRBIJA

CFS

CFS. JOURNAL OF COMPUTER AND FORENSIC SCIENCES

Belgrade, 2025

PUBLISHER

University of Criminal Investigation and Police Studies, Cara Dušana 196, 11080 Belgrade, Serbia

TABLE OF CONTENTS

Journal of COMPUTER AND FORENSIC SCIENCES

CFS 2025, Vol. 4, Issue 1, pp. 1
Editorial

EDITORIAL

Further down the Road of Machine Learning

Milan Gnjatović
Editor-in-Chief
University of Criminal Investigation and Police Studies, Belgrade;
milan.gnjatovic@kpu.edu.rs
Published: September 17, 2025

In this issue of the Journal of Computer and Forensic Sciences, we present three interesting articles in the field of machine learning.

The first article [1] reports on a comparative analysis of machine learning models for real-time object detection.

The second article [2] explores the development of a heart rate estimation system that integrates wearable sensors with machine learning techniques.

The last article [3] explores the application of time series algorithms to enhance anomaly detection in a cybersecurity context.

I thank all authors and reviewers for supporting us in this indifferent universe of academic publishing. Your support is much appreciated!

REFERENCES

[1] M. Wittenhagen, "Comparative Analysis of Machine Learning Models for Real-Time Object Detection", Journal of Computer and Forensic Sciences, vol. 4, no. 1, pp. 3–19, 2025.

[2] N. Kumar, D. Kumar, "Heart Rate Estimation Using Wearable Sensors and Machine Learning", Journal of Computer and Forensic Sciences, vol. 4, no. 1, pp. 20–31, 2025.

[3] M. M. Živanović, M. D. Milošević, E. Kisić, "Application of Time Series Algorithms in Cybersecurity", Journal of Computer and Forensic Sciences, vol. 4, no. 1, pp. 32–61, 2025.

ORIGINAL RESEARCH PAPERS

Journal of COMPUTER AND FORENSIC SCIENCES

CFS 2025, Vol. 4, Issue 1, pp. 3-19
https://doi.org/10.5937/jcfs4-58032
Original research paper

Comparative Analysis of Machine Learning Models for Real-time Object Detection

Merlin Wittenhagen[1*]

[1] Student, School of Engineering and Architecture, SRH University, Heidelberg, Germany

* Corresponding author: merlinwittenhagen@icloud.com

Received: April 6, 2025 • Accepted: May 6, 2025 • Published: June 20, 2025

Abstract: Object detection is a fundamental task in computer vision with applications ranging from autonomous driving to industrial automation and medical imaging. This report presents a comparative analysis of six well-known object detection models: three small models for edge computing and three large models likely more suited for usage on high-performance systems. The models YOLOv10-Nano, MobileNetV3-SSDLite, EfficientDet-D0, Faster R-CNN, YOLOv10-Large, and DETR were evaluated and compared based on their performance in terms of inference speed, accuracy, and computational efficiency. The evaluation is conducted through both literature-based benchmarks and empirical tests on two different systems: an Apple Silicon M1 Pro-based system and an NVIDIA RTX 3080Ti-powered computer. Results show that YOLOv10 models consistently outperform the other models in real-time object detection as well as achieving superior accuracy in general while maintaining significantly lower inference times. The analysis further highlights compatibility issues with certain hardware, particularly focusing on PyTorch's MPS backend on Apple Silicon, which leads to serious performance drops in some models. The findings highlight the importance of choosing the right model and appropriate hardware for specific application scenarios.

Keywords: object detection; model benchmarking; inference efficiency, hardware-aware evaluation.

1. INTRODUCTION

Object detection has become one of the most important tasks in computer vision, with applications ranging from autonomous driving to medical imaging and automation in industrial facilities. While some use cases allow for complex object detection models with high computational demands, others are constrained by limited system capabilities (edge computing or mobile devices), requiring more efficient solutions.

Over the last years, multiple object detection models have been developed and evolved, each having unique advantages and tradeoffs. These can be speed, accuracy, and computational cost. Besides common CNN-based architectures like YOLO and Faster R-CNN, which have been the most obvious choice for most problems in recent years, new Transformer-based models such as DETR (DEtection TRansformer) were developed to improve object detection in complex scenes.

In the scope of this comparative analysis, six well-known object detection models – YOLOv10nano, MobileNetV3-SSD Lite, EfficientDet-D0, Faster R-CNN, YOLOv8-large, and DETR – will be evaluated to determine their suitability for various applications. This will be done by comparing performance metrics like inference speed, accuracy, computational efficiency, and their applicability in edge or high-performance environments.

The primary aim of this work is to give insights into how these models perform under various constraints, such as limited computational power compared to high-performance setups. In addition to that, it will be highlighted whether Transformer-based models like DETR can compete with or surpass more traditional CNN-based approaches. The paper aims to provide information on which model is best for certain use cases, keeping metrics like accuracy, computational demand, and speed in mind.

2. MODEL OVERVIEW

2.1. YOLO

For this work, two sub models of the YOLO-model (You Only Look Once) were used. In fact, the chosen models are based on the tenth generation of YOLO, which is the result of improvements in performance over the years. The following subsection will give an insight into the general function of earlier YOLO models as well as YOLOv10.

2.1.1. YOLO – Basic Functionality

As a single-stage model, YOLO processes the entire image in one step, simultaneously predicting class labels and bounding boxes. This sets it apart from other models that use a two-stage approach (like Faster R-CNN, which is explained later), which works by proposing regions and then classifying an object separately [1, 42].

The model divides an input image into a grid of S x S dimensions, where each grid cell is detecting if an object lies within that cell. Each cell then predicts a set number of bounding boxes, which are described by coordinates (x, y, w, h), representing the position of the bounding box as well as its dimensions relative to the image size [2]. Those bounding boxes are then getting assigned a confidence score, which combines the probability of an object being present and the accuracy of the bounding box. This accuracy is measured by Intersection over Union (IoU). In addition to that, the network also predicts class probabilities to identify the detected object [2].

After calculating all confidence scores and bounding boxes, a threshold is applied to filter out predictions with low confidence. In previous YOLO versions (v1-v8), the Non-Maximum Suppression (NMS) was used to remove needless bounding boxes by only selecting the ones with the highest confidence scores while preventing keeping strongly overlapping boxes based on the Intersection over Union metric [3, 42].

With the introduction of YOLOv10, NMS was dropped to further increase computational efficiency. This was done by optimizing processes during training, which allows the model to learn to predict only the most accurate bounding boxes directly [4]. For final detection

(classification and locating the bounding boxes), three different CNNs are used. They are optimized to detect large, medium, and small objects in an image [6]. The general architecture of YOLOv10 is shown in Figure 1.

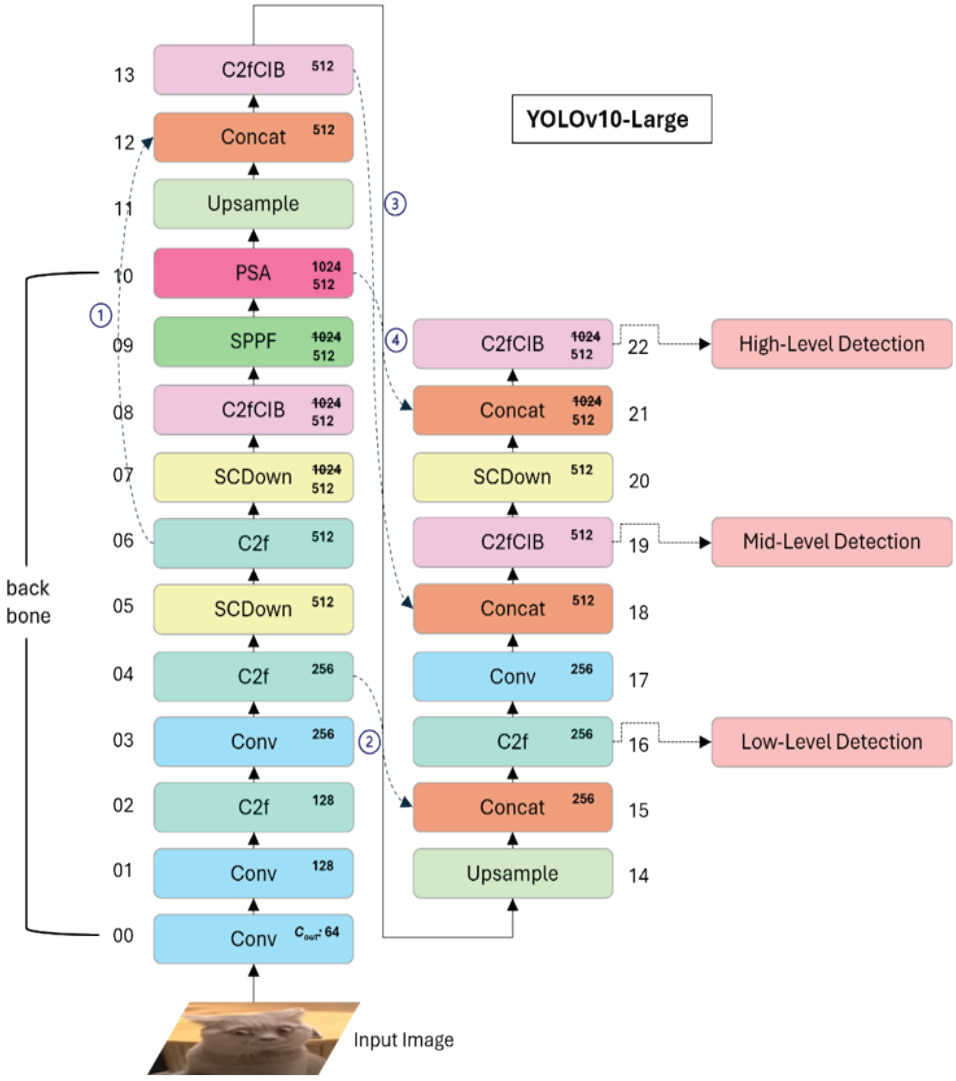

Figure 1. *Architecture of YOLOv10. Adapted from [7].*

2.2. Faster R-CNN – Basic Functionality

Other than YOLO, Faster R-CNN (FRCNN) is a two-stage object detection model, which separates the process of localizing objects and classifying them. To achieve this, FRCNN utilizes a Region Proposal Network (RPN), which is responsible for accurately predicting the regions where objects are likely to be present. These region proposals are then used to generate bounding boxes. The second stage of the pipeline classifies the object within the identified region and refines the bounding box for more precise localization [8]. In detail,

the model processes an input image through a CNN backbone (in this case, ResNet-101) to extract feature maps. These feature maps are passed into the RPN, which generates region proposals that likely contain objects. Similar to earlier YOLO models, the RPN assigns anchor boxes to different locations in the image and predicts whether each anchor contains an object or not. The bounding boxes are further refined to better frame the detected object [8].

Once the RPN generates a set of region proposals, they undergo Region of Interest (RoI) Pooling, which ensures that all proposals are transformed into a fixed-size feature representation. These feature maps serve as the input for a fully connected layer, which performs object classification and further adjusts the bounding box for more precise localization [9].

After object classification and bounding box refinement, a confidence threshold is applied to filter out low-confidence predictions. To avoid multiple overlapping bounding boxes for the same object, Non-Maximum Suppression (NMS), as described in the previous chapter, is applied [8]. Figure 2 shows a rough depiction of the Faster R-CNN model.

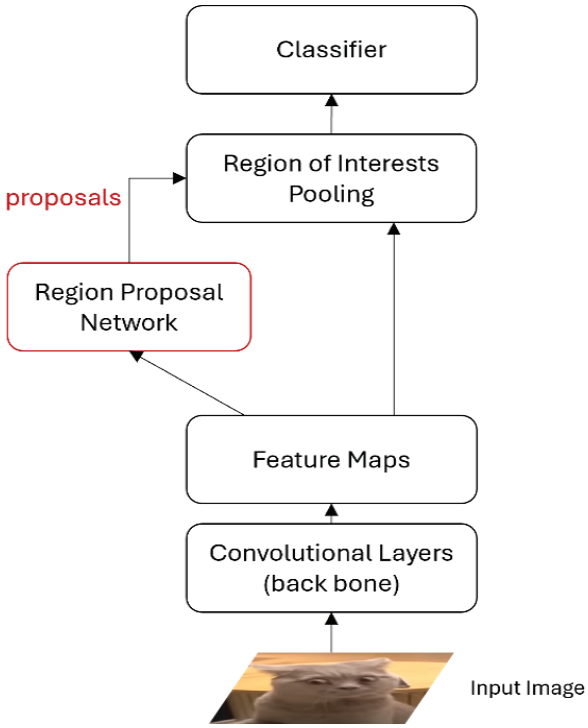

Figure 2. *Basic Architecture of Faster R-CNN.* Adjusted from [10].

2.3. MobileNetV3 – Basic Functionality

The MobileNetV3 model that was chosen for this work is the SSDlite320 MobileNetV3-Large. It is a lightweight object detection architecture that combines the MobileNetV-Large backbone with the so-called Single Shot MultiBox Detector (SSD) framework. This setup was designed to deliver high-performance object detection suitable for mobile

and embedded systems. MobileNetV3 is a convolutional neural network architecture that is optimized for mobile applications. It uses different techniques to balance accuracy and latency effectively. The "Large" variant is adapted for scenarios requiring higher accuracy while remaining computationally efficient [11]. The SSD makes the combined model a single-stage object detector like YOLO. It does not rely on a separate RPN to propose regions. Instead, the SSD predicts object classes and bounding boxes directly from the feature maps extracted from the MobileNetV3 CNN [12, 13].

2.4. EfficientDet-D0 – Basic Functionality

EfficientDet-D0 is also an efficient, lightweight object detection model. Like other models of its kind, it aims to optimize the trade-off between accuracy and computational efficiency to make it suitable for mobile applications. Unlike traditional object detection architectures that scale only in depth or width, EfficientDet-D0 uses compound scaling, which scales the backbone network, the feature fusion layers, and the prediction heads to improve overall efficiency [14]. EfficientDet combines a classic CNN architecture with a Bidirectional Feature Pyramid Network (BiFPN) to efficiently fuse multi-scale features. This way the feature extraction as well as the prediction heads are optimized, which provides a balanced trade-off between computational cost and accuracy. [14] Figure 3 shows a simplified version of the EfficientDet network, which utilizes the EfficientNet backbone (CNN) and extracts features from multiple layers to feed them into the BiFPN. The output of the BiFPN is then used as input for two fully connected layers for class prediction and box prediction [14].

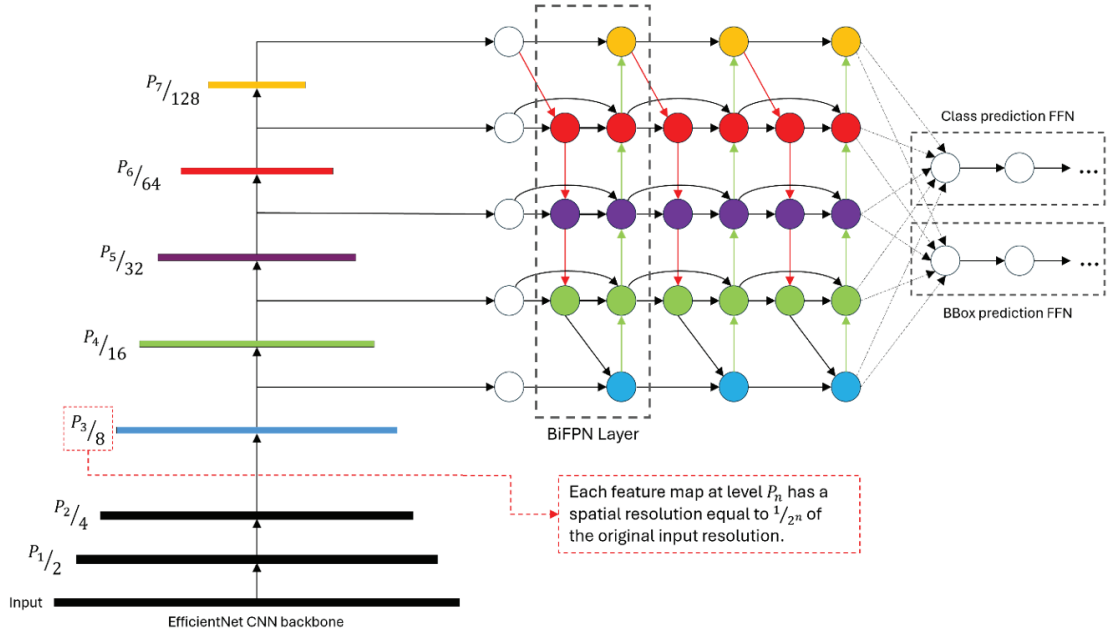

Figure 3. *Architecture of EffiecientDet-D0. Adjusted from [15].*

2.5. DETR – Basic Functionality

Contrary to the models described before, the DETR (DEtection TRansformer), as the name states, uses a Transformer architecture to interpret features extracted by a CNN. Like Faster R-CNN, the backbone used to extract features from input images is typically ResNet-50 or ResNet-101. The generated feature maps are used as input for the Transformer Encoder. The Transformer Encoder processes the extracted feature maps and learns long-range dependencies between different regions of the image. Unlike other models that generate a fixed or dynamic number of regional proposals or divide the image into a grid, like YOLO does, DETR uses a fixed number of learnable object queries. Each of those queries represents a potential object in the image, even if some queries remain unused since no object is detected. Those queries are in general tensors with n dimensions, which are initialized randomly and then are optimized during the training phase of the model. There could be, for example, 100 query tensors, which all search for a different object in the feature map (the amount of object queries determines the number of detectable objects). These queries are processed by the Transformer Decoder, which relates them to the encoded feature map. Since each query focuses on a specific region in the image, the model is able to determine the location of an object and its class. The final predictions are technically done by using two separate prediction heads in the form of two different fully connected layers to determine the class and the bounding box of the object [16, 17, 5].

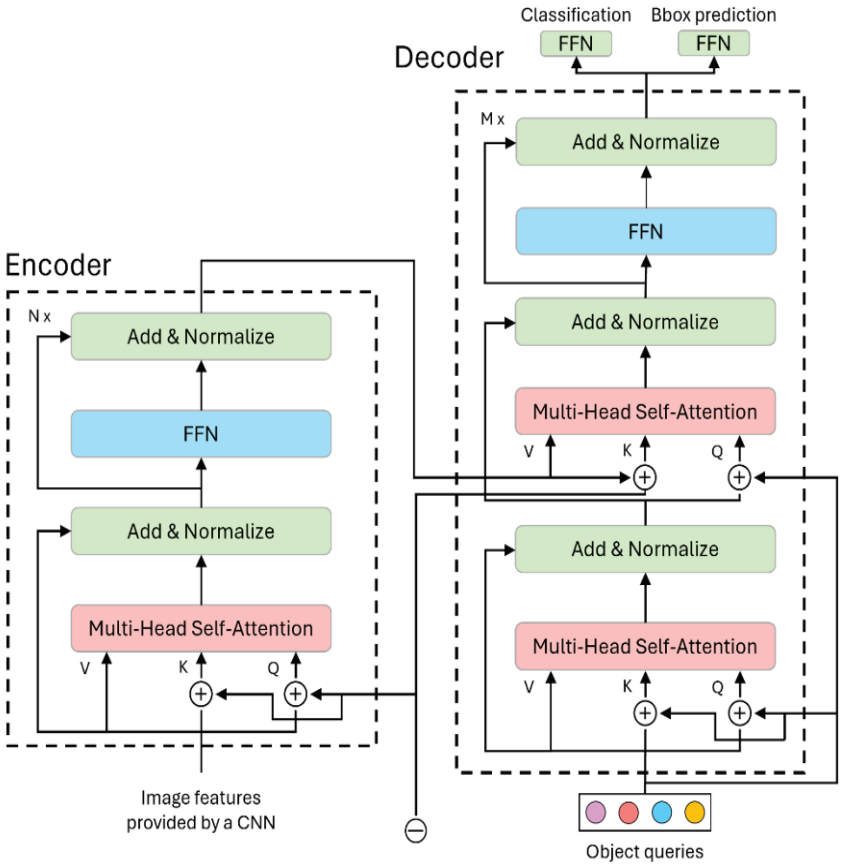

Figure 4. *Encoder-Decoder Architecture of DETR. Adapted from [18].*

3. METHODS

3.1. Comparison of the Models Based on Existing Evaluation Data

The first comparison was made by choosing three low-size and high-speed as well as three high-size and relatively low-speed models, which still can be used for real-time processing. Chosen were popular models that are widely used in the realm of object detection. The six models that were described in the previous chapter were then compared based on their number of parameters, their computational cost in FLOPS (Floating Point Operations per Second), the model size in terms of storage demand, the inference time in milliseconds, and the accuracy based on the mAP@0.5:0.95 (Mean Average Precision across the defined range of Intersection over Union, detailed explanation in the chapter "Explanation of the Mean Average Precision"). The models were compared overall and with respect to their respective aim (separated by small and large models).

The information about the researched models was taken from different sources, since there is no comparison between all of them. This means that the evaluation results for each model were probably obtained on a different system. To still provide useful insight, it was made sure that all models were trained and evaluated based on the 2017 COCO dataset [19].

3.2. Testing the Models on Different Systems

To give an insight into the performance of the tested models when run on systems that are more likely to be found in a real-word application, the performance in terms of inference time was measured on two different devices. The computers used were chosen because of their availability while conducting this work.

The first system was an Apple MacBook Pro with an ARM64-based M1 Pro chip. This chip consists of 8 performance and 2 efficiency CPU cores working at 2.06-3.22 GHz as well as 16 GPU cores. The system uses a unified memory of 16 GB [20]. In addition to that, the chip inherits a 16-core Apple Neural Engine, which is technically an NPU (Neural Processing Unit), which is optimized for convolutions and matrix multiplications (tensor operations). However, the latter was not used since the tested models were all based on the PyTorch framework, which does not support the ANE (Apple Neural Engine). To still run the models efficiently, Apple's MPS (Metal Performance Shaders) API was used to run them on the GPU. PyTorch contained MPS backend support since version 1.12 while still being unable to offer the same functionality as other APIs, which results in poorer performance [21].

As a second test system, a more "standard" tower PC was used. It operates on the more common 64-bit x64-CPU-architecture. The CPU was an Intel Core i7 12700k with a total of 12 cores (8 performance, 4 efficiency) running at 3.6-4.9 and 2.7-3.8 GHz [22]. The system memory consisted of 32 GB DDR4 at 4000 MHz. These specifications were only included in this description to be precise and transparent. Their influence would be marginal since the models were run exclusively on the GPU. The used GPU was an NVIDIA

GeForce RTX 3080Ti with 12 GB of video memory and 10240 CUDA Cores, which makes it very suitable for AI and ML related tensor/matrix calculations [23].

The two systems differ significantly in their intended application purpose, as one is optimized for low energy consumption mobile use and the other is set up for energy-intensive, high-performance operation. This comparison was done to show how insightful commonly available information on the performance of object detection models really is, since most systems are not able to match the performance of benchmark systems. This comparison aimed to show how much the inference times actually differ from the proposed benchmark values generated with high-end systems if less powerful hardware is used. In addition to that, this was intended to also highlight possible compatibility issues with certain hardware, which could cause an object detection model to become unusable on a device.

3.3. Description of the Mean Average Precision

The mean Average Precision as introduced in the COCO benchmark [43] was used to evaluate the performance of the models in terms of box prediction and object-classification accuracy. This chapter will describe how the mAP is calculated and which subtypes are commonly used to evaluate object-detection models. In general, the mAP is dependent on the Intersection over Unit, which is then used to calculate the share of correctly detected boxes based on precision and recall. It is used as a metric to describe the robustness of an object-detection model.

3.3.1. Intersection Over Unit (IoU)

The IoU describes the accuracy of a predicted bounding box relative to the actual box (Ground Truth). This is calculated by dividing the area of overlap between the predicted and the actual box by the area of their union [24]. This can be mathematically described as:

$$IoU = \frac{|A \cap B|}{|A \cup B|} = \frac{|I|}{|U|}$$

1where:

- *IoU* is the intersection over unit,
- *A* is the area of the first box,
- *B* is the area of the second box.

The expected result is a number between zero and one, with higher values showing a higher overlap and therefore a better result. A graphical example can be seen in Figure 5. Precision and Recall are used to describe the portion of correctly predicted boxes relative to the total number of predictions and to describe the portion of correctly predicted boxes relative to the total number of existing boxes.

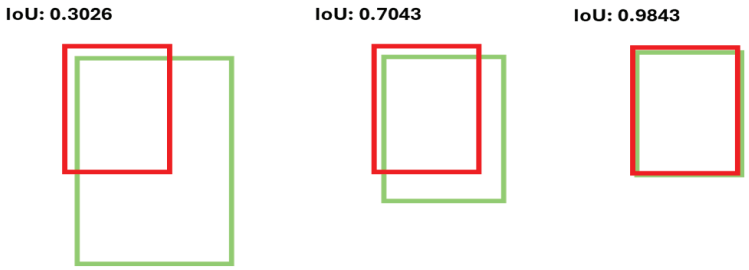

Figure 5. *Example – IoU Bounding Boxes. Adjusted from [25].*

3.3.2. Calculation of the mAP

As mentioned in the introduction to this chapter, there are multiple versions of the mAP. In general, the AP (Average Precision) is the area under the precision-recall curve. Earlier object detection models were evaluated using the mAP@0.5 metric, where a bounding box is rated as true positive if the IoU is equal to or larger than the threshold of 0.5. The downside of a fixed threshold of 0.5 is that it evaluates rather poor predictions as true positives, even if the bounding box is relatively far off. To get a more robust way of evaluating the performance of an object-detection model, the mAP@0.5:0.95 is used. It describes the mean Average Precision over a range of 10 thresholds between 0.5 and 0.95. This shows not only the model's capability of detecting an object, but it also gives insight into the ability to find accurate bounding boxes [26].

The final equations used to calculate the mAP@0.5 and the mAP@0.5:0.95 are shown in Equation 2 and Equation 3.

To make the comparison between the six selected models, only the mAP@0.5:95 was chosen as the accuracy metric since it provides the more meaningful insight. This method is currently the most used for the evaluation of object-detection models, often as a benchmark metric after training and evaluation with the COCO data set [27]. Therefore, the mAP@0.5 was not used as a metric for the comparison conducted for this paper.

$$mAP@0.5 = \frac{1}{N}\sum_{c=1}^{N} AP_c@0.5$$

2where:

- $mAP@0.5$ is the mean Average Precision at an IoU threshold of 0.5,
- N is the total number of object classes,
- $AP_c@0.5$ is the Average Precision for class c at an IoU threshold of 0.5.

$$\text{mAP@0.5:0.95} = \frac{1}{N}\sum_{c=1}^{N}\left(\frac{1}{10}\sum_{k=1}^{10}\text{AP}_c(IoU_k)\right)$$

3where:

- *mAP*@0.5:0.95 is the mean Average Precision, averaged over 10 *IoU* thresholds from 0.5 to 0.95 in steps of 0.05,
- *N* is the total number of object classes,
- $AP_c(IoU_k)$ is the Average Precision for class *c* at a specific *IoU* threshold IoU_k,
- *k* represents the index for the *IoU* thresholds, where $IoU_k \in \{0.5, 0.55, 0.6, \ldots, 0.95\}$,
- $\frac{1}{10}$ ensures the average over the 10 *IoU* thresholds.

4. RESULTS

4.1. Comparison Results Based on Available Evaluation Data

Comparing the models based on already available data first of all shows the clear differences in light models for edge computing and heavy models for high-performance applications. One of the key factors is the size of the model in terms of parameters and total file size. The small models range from 2.3 to 3.9 million parameters, while the large models have around ten times as many. The small models, namely the YOLOv10-Nano, the MobileNetV3-SSDLite, and the EfficientDet-D0, are showing the typical trade-off between low computational demand, low inference, and reasonable accuracy. YOLOv10-Nano has the lowest inference time of all models, with the highest accuracy of the smaller models based on the mAP@0.5:0.95, while requiring more FLOPS than other fast models. This seems counterintuitive since more FLOPS mean more calculations, which should increase the inference time. In the specific case of YOLOv10, this seems not to be an issue. This is because YOLOv10's architecture is highly optimized for parallel calculations on modern hardware platforms, which makes it faster than other models with less computational demand in terms of FLOPS [2]. For the large models where Faster R-CNN, YOLOv10-Large, and DETR were compared, it is also noticeable that YOLOv10 outperforms its competitors in terms of speed, accuracy, and memory demand. In this particular case, even the computational demand is with around 120 GFLOPS lower than that of Faster R-CNN and DETR, with 134 and 180 GFLOPS. The YOLOv10-Large model is by far the most accurate of all six models, outperforming the second best (DETR) by roughly ten percent points (53.3 % vs 42 %). Even the YOLOv10-Nano model outperforms every compared model excluding the DETR and YOLOv10-Large at around 39.5 % accuracy.

Based on the provided data, YOLOv10-Nano is the best small model for edge computing and mobile applications. It is by far the fastest model, provides the highest accuracy, and has the smallest memory demand, which makes it more suitable for devices with limited memory.

The result for the large models is similar. YOLOv10-Large outperforms Faster R-CNN and DETR in every point. It is faster, more accurate and needs less memory as well as less computational power. In terms of real-time detection capabilities, it even competes with the small models, having an inference of around 7.28 ms.

To conclude the data interpretation, it is apparent that the YOLOv10 models are at the moment the best choice to solve object detection problems, especially when it comes to

real-time detection. Both the Nano and the Large model outperform their competitors in almost every metric, which leads to the stated conclusion.

The comparison data in Table 1 was obtained from the following sources: [4, 28, 29, 30, 31, 32, 33, and 34]. The table was also divided into small models and large models, as can be observed.

Table 1. *Comparison of Object Detection Models.*

Model	Params (Million)	FLOPs (GFLOPs)	Size (MB)	Inference Time (ms)	Accuracy (mAP@0.5:0.95)
YOLOv10-Nano	2.3	6.7	4	1.56	39.5 %
MobileNetV3-SSDLite	3.9	0.55	5	3.5	39 %
EfficientDet-D0	3.9	2.54	16	3.92	34.6 %
Faster R-CNN (ResNet-101)	41.8	134.4	160	54	37 %
YOLOv10-Large	24.4	120.3	100	7.28	53.3 %
DETR (ResNet-50)	41.6	180	159	36	42 %

Rounded values

4.2. Performance Results on Different Systems

Running the models on the two selected systems resulted in the inference times displayed in Table 2. Like before, the table is split into small and large models. Looking at the M1 Pro results for the small models – YOLOv10-Nano, MobileNetV3-SSDLite, EfficientDet-D0 – a very large discrepancy in inference times becomes apparent. The YOLO-Nano model is by far the fastest of the three, with an average inference time of around 17 ms, while the EfficientDet-D0 model comes second with an average inference of 70 ms. The third small model, however, did not perform well at all, processing the images with an average inference of roughly 760 ms, which is significantly worse than the other two models.

Looking at the M1 Pro results for the large models – Faster R-CNN, YOLOv10-Large, DETR – the Large YOLO model also shows the best average inference, with around 37 ms. DETR performed well for a large model at 140 ms. Faster R-CNN, however, performed so significantly badly that it can be deemed unusable on this platform. It has an average inference of circa 100,000 ms, which is extremely high. The reason for this behavior will be discussed in detail, since it highlights a significant flaw of using PyTorch models on some devices.

Coming to the results generated with the RTX 3080Ti and first looking at the performance of the small models, each can be evaluated as working properly. The YOLOv10-Nano – like on the M1 Pro – performed the best with a noticeable lead over the other two models. With an average inference of ~3.5 ms, it runs more than seven times faster than MobileNet with ~25 ms and almost six times faster than EfficientDet with ~20 ms inference.

The results for the large models run on the 3080Ti show that YOLOv10-Large also performed the best in terms of speed. It only needed roughly 7.3 ms to process an image, while

Faster R-CNN needed 36 ms and DETR 26.5 ms on average. This time, Faster R-CNN worked as intended and performed rather well compared to the test on the M1 Pro.

Concluding the results of the system comparison, it is possible to say that in terms of speed, both YOLO-models outperformed all other four models regardless of being compared to the small or large ones. This is especially interesting since YOLOv10-Large not only outperforms MobileNet and EfficientDet at raw speed, it also tremendously outperforms them in terms of accuracy, as can be seen in Table 1. The results reinforce the statement that, out of the compared models, YOLOv10 currently seems to be the best choice for small and large models in terms of speed or, more precisely, for real-time object detection.

Table 2. *Average Inference Times on M1 Pro and Nvidia RTX 3080Ti.*

Model	Apple Silicon M1 Pro	Nvidia RTX 3080Ti
YOLOv10-Nano	17	3.5
MobileNetV3-SSDLite	760	25
EfficientDet-D0	70	20
Faster R-CNN (ResNet-101)	100000	36
YOLOv10-Large	37	7.3
DETR (ResNet-50)	140	26.5

Average values in milliseconds

5. DISCUSSION

5.1. Interpretation of the Results

If it comes to real-time object detection, out of the tested models there is currently no better choice than YOLOv10+ (v10 or newer). This is due to its described lightweight architecture with vast optimizations to allow for fast calculations resulting in very low inference times while providing benchmark setting accuracy.

Looking a bit further and taking more models into account which were not compared in this paper, depending on the task there are serious competitors for YOLOv10 and newer versions. For real-time detection one alternative would be RTDETR (Real-Time DEtection TRansformer) which offers comparable performance. It could be further researched, if this model even has some distinct advantages over YOLO [35].

To not give the impression that YOLOv10+ is always the best choice, it does not set the bar when it comes to high-precision-detection models and zero-shot object detection models. In the realm of high precision models YOLOv10+ can be used but there are better alternatives, namely Co-DETR (Check out DETR) and DETA (Detection Transformers with Assignment) [36]. Both models provide an mAP@0.5:0.95 higher than 62 [36]. Regarding zero-shot object detection (detection objects without being explicitly trained on those classes) there are significantly better models than YOLOv10+. If such a problem must be solved, Grounding Dino 1.5 Pro or OWLv2 should be taken as possible alternatives [36]. As always, it should be researched which model would provide the best performance in terms of accuracy and speed for a specific task.

5.2. Compatibility and Performance Issues

Looking at the Results chapter, and especially at the results for the average inference times of the M1 Pro tests, the bad results for the MobileNet and Faster R-CNN tests should raise some questions. The first should be why the inference times are significantly higher than expected and why it only happened on the M1 Pro and not on the RTX 3080Ti.

As already mentioned before, there is MPS backend support in PyTorch. Unfortunately, there are still some undisclosed issues where some operations are not yet implemented using MPS. If this happens and the CPU fallback (utilizing the CPU instead of the GPU) does not work, it could result in such high inference times as observed in the case of Faster R-CNN. These problems are widely reported and will probably be solved in the future [37, 21]. One possible alternative would be to search for TensorFlow-based models, as the MPS backend support of TF is more refined than that of PyTorch since it is not in the beta phase anymore [21, 38].

Issues like this are less likely to occur on NVIDIA GPUs, as most machine learning frameworks are built around NVIDIA's CUDA (Compute Unified Device Architecture) API [39, 40]. Deploying AI models in Edge Computing across different hardware and software ecosystems can impose new hurdles that need to be recognized and overcome. Model size, computational demand, and format can be some of them, especially when running AI models on not optimized platforms like Raspberry Pi, Arduino, or STM32, since they do not have hardware acceleration or suitable processors at all. Luckily there are some specially developed Edge Computing solutions like NVIDIA Jetson, which, for example, supports CUDA, PyTorch and is optimized for deep learning, using special Tensor-Core GPUs or ARM-based chips [41]. Finally, it is always necessary to choose the right model for the right task and to make sure that the system used does support such a model, as the conducted tests showed.

6. CONCLUSION

The comparison showed that out of the selected models, YOLOv10 performed the best with regard to every metric. That goes for the Nano version in the realm of small models and for the large version regarding the other tested large models. Additional to this, an inference test and comparison showed how high the inference is likely to be if the models are run on more standard devices and not on high-end benchmark systems. It was proved that some of the models provided very satisfying performance in terms of accuracy and speed on a low-energy-consumption laptop and a relatively high-end computer. If run on powerful hardware, all tested models could be used for real time-object detection with acceptable inference times or frames per second. If run on a lower performance system, only the YOLO models and – with significantly less performance – the EfficientDet-D0 could be used for real-time detection. The issues regarding the use of PyTorch with the Apple MPS API made it obvious that not every model is suited for every device, even if the technical specifications match on the first glance.

To finish this paper, it should be mentioned that object detection models are still evolving fast, and the insights gained in the scope of this work might be obsolete in just a matter

of time. Also worth mentioning is the diverse use of different approaches to solve quite similar problems, which leaves room for improvement and new ideas.

FUNDING
This research received no external funding.

INSTITUTIONAL REVIEW BOARD STATEMENT
Not applicable.

INFORMED CONSENT STATEMENT
Not applicable.

CONFLICTS OF INTEREST
The author declares no conflict of interest.

REFERENCES

[1] R. Kundu, "YOLO: Algorithm for object detection explained [+examples]," V7 Labs Blog, 2023.

[2] DataCamp, "YOLO object detection explained: A beginner's guide," DataCamp Blog, 2024.

[3] Ultralytics, "Nicht-maximum-unterdru¨ckung (nms)," Ultralytics Glossar, 2024.

[4] Ultralytics, "Yolov10: End-to-end-objekterkennung in echtzeit," Ultralytics YOLO Docs, 2024.

[5] H. Face, "Deformable DETR: Deformable transformers for end-to-end object detection." https://huggingface.co/docs/transformers/en/model_doc/deformable_detr, 2025. Accessed: 2025-03-11.

[6] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, and G. Ding, "YOLOv10: Real-time end-to-end object detection." https://github.com/THU-MIG/yolov10/blob/main/ultralytics/nn/modules/head.py, 2024. Accessed: 2025-03-11.

[7] d4r6j, "YOLOv10: Model review." https://velog.io/@d4r6j/YOLOv10-1.-Model-Review, 2024. Accessed: 2025-03-11.

[8] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," arXiv preprint arXiv:1506.01497, 2015.

[9] R. B. Girshick, "Fast R-CNN," in Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 1440–1448, 2015.

[10] R. User, "Network structure diagram of Faster R-CNN." https://www.researchgate.net/ fig-ure/Network-structure-diagram-of-Faster-R-CNN-Faster-R-CNN-is-mainly-divided-into-the_ fig1_341871095. Accessed: 2025-03-07.

[11] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for MobileNetV3," arXiv preprint arXiv:1905.02244, 2019.

[12] V. Vryniotis, "Everything you need to know about Torchvision's SSDlite implementation." https://pytorch.org/blog/torchvision-ssdlite-implementation/, 2021. Accessed: 2025-03-07.

[13] P. Team, "SSDlite implementation in Torchvision." https://github.com/pytorch/vision/blob/main/torchvision/models/detection/ssdlite.py, 2025. Accessed: 2025-03-07.

[14] M. Tan, R. Pang, and Q. V. Le, "EfficientDeT: Scalable and efficient object detection," in Proceed-ings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 10781–10790, 2020.

[15] D. Buongiorno, D. Caramia, L. D. Ruscio, and A. Brunetti, "EfficientDet-D0 architecture: Effi-cientNet-B0 as backbone network with multiple BiFPN layers." https://www.researchgate.net/figure/EfficientDet-D0-architecture-EfficientNet-B0-34-is-the-backbone-network-multiple_ fig2_365439360, 2022. Accessed: 2025-03-07.

[16] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in European Conference on Computer Vision (ECCV), pp. 213–229, 2020.

[17] H. Face, "DETR: End-to-end object detection with transformers." https://huggingface.co/ docs/transformers/en/model_doc/detr, 2025. Accessed: 2025-03-11.

[18] G. Boesch, "DETR: End-to-end object detection with transformers." https://viso.ai/ deep-learning/detr-end-to-end-object-detection-with-transformers/, 2024. Accessed: 2025-03-11.

[19] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dol-lár, and C. L. Zitnick, "Microsoft COCO: Common objects in context." https: //cocodataset.org/#download, 2014. Accessed: 2025-03-07.

[20] Notebookcheck, "Apple M1 Pro prozessor – benchmarks und specs," 2021.

[21] A. Inc., "Accelerated PyTorch training on mac." https://developer.apple.com/metal/pytorch/, 2025. Accessed: 2025-03-10.

[22] I. Corporation, "Intel® core™ i7-12700k prozessor (25 mb cache, bis zu 5,00 ghz) spezifikationen." https://www.intel.de/content/www/de/de/products/sku/134594/intel-core-i712700k-processor-25m-cache-up-to-5-00-ghz/specifications.html, 2021. Zugegriffen: 2025-03-10.

[23] NVIDIA, "GeForce RTX 3080 and RTX 3080 Ti graphics cards," 2024.

[24] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," Proceedings of the IEEE/CVF Con-ference on Computer Vision and Pattern Recognition (CVPR), pp. 658–666, 2019.

[25] A. Rosebrock, "Intersection over union (IoU) for object detection." https://pyimagesearch. com/2016/11/07/intersection-over-union-iou-for-object-detection/, 2016. Accessed: 2025-03-10.

[26] D. Shah, "Mean average precision (mAP) explained: Everything you need to know." https: //www.v7labs.com/blog/mean-average-precision, 2022. Accessed: 2025-03-10.

[27] C. Consortium, "COCO object detection evaluation." https://cocodataset.org/ #detection-eval, 2025. Accessed: 2025-03-10.

[28] Ultralytics, "YOLOv10 vs EfficientDet comparison." https://docs.ultralytics.com/com-pare/yolov10-vs-efficientdet/, 2024. Accessed: 2025-03-09.

[29] Ultralytics, "YOLOv10 model comparisons." https://docs.ultralytics.com/de/models/yolov10/#comparisons, 2024. Accessed: 2025-03-09.

[30] P. Team, "MobileNetV3 small model." https://pytorch.org/vision/main/models/ generat-ed/torchvision.models.mobilenet_v3_small.html#torchvision.models. mobilenet_v3_small, 2025. Accessed: 2025-03-09.

[31] O. Toolkit, "EfficientDet-D0 TensorFlow model." https://github.com/openvinotoolkit/open_model_zoo/blob/master/models/public/efficientdet-d0-tf/README.md, 2025. Accessed: 2025-03-09.

[32] P. Team, "Faster R-CNN with ResNet-50 FPN." https://pytorch.org/vision/main/models/generat-ed/torchvision.models.detection.fasterrcnn_resnet50_fpn.html, 2025. Accessed: 2025-03-09.

[33] F. Research, "DETR: End-to-end object detection with transformers." https://github.com/ face-bookresearch/detr/blob/main/README.md, 2025. Accessed: 2025-03-09.

[34] PromptLayer, "DETR ResNet-50 model overview." https://www.promptlayer.com/models/ detr-resnet-50-6671, 2025. Accessed: 2025-03-09.

[35] Ultralytics, "RTDETRv2 vs YOLOv10: A technical comparison for object detection." https:// docs.ultralytics.com/compare/rtdetr-vs-yolov10/, 2024. Accessed: 2025-03-10.

[36] A. Kouidri, "Top object detection models in 2024." https://www.ikomia.ai/blog/top-object-detection-models-review, 2024. Accessed: 2025-03-10.

[37] PyTorch Community, "Metal performance shader (MPS) discussion forum." https://dis-cuss.pytorch.org/c/metal-performance-shader/38, 2025. Accessed: 2025-03-10.

[38] A. Inc., "Accelerated TensorFlow training with metal." https://developer.apple.com/met-al/tensorflow-plugin/, 2025. Accessed: 2025-03-10.

[39] R. Kumar, "List of CUDA-aware frameworks in machine learning." https://www. devopss-chool.com/blog/list-of-cuda-aware-framework-in-machine-learning/, 2024. Accessed: 2025-03-10.

[40] N. Corporation, "Deep learning software." https://developer.nvidia.com/ deep-learning-software, 2025. Accessed: 2025-03-10.

[41]N. Corporation, “Eingebettete systeme: Entwicklerkits und module von nvidia jetson.” https: //www.nvidia.com/de-de/autonomous-machines/embedded-systems/, 2025. Zugegriffen: 2025-03-10.

[42]J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time ob-ject detection,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recogni-tion (CVPR), pp. 779–788, 2016.

[43]T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Mi-crosoft COCO: Common objects in context,” in European Conference on Computer Vision (ECCV), pp. 740–755, Springer, 2014.

Journal of COMPUTER AND FORENSIC SCIENCES

CFS 2025, Vol. 4, Issue 1, pp. 20-31
https://doi.org/10.5937/jcfs4-58752
Original research paper

Heart Rate Estimation Using Wearable Sensors and Machine Learning

Nikhil Kumar[1] and Darshan Kumar[2]

[1] Octilyon; nikhil.kumar@octilyon.com

[2] Octilyon; darshan.kumar@octilyon.com

* Corresponding author: nikhil.kumar@octilyon.com

Received: May 9, 2025 • Accepted: June 8, 2025 • Published: June 20, 2025

Abstract: This research explores the development of a heart rate estimation system that integrates wearable sensors with machine learning techniques to achieve high accuracy, low cost, and real-time performance. The project aims to build two-stage phases: a software pipeline for model training and a hardware framework for real-world testing. In the first phase, various machine learning algorithms are trained and fine-tuned using the publicly available PPG DaLiA dataset, which contains physiological data collected during everyday activities. The training process focuses on optimizing performance across different model architectures and configurations. The second phase involves implementing the trained model on a real-time embedded system. An ESP32 microcontroller serves as the central unit to collect data from multiple sensors, including electrocardiography (ECG), photoplethysmography (PPG), galvanic skin response (GSR), temperature, and a 3-axis accelerometer. This data is transmitted wirelessly for preprocessing and inference. The user will see their final predicted heart rate on both an OLED display and a user interface (UI) dashboard.

Keywords: heart rate; PPG; ECG; GSR; temperature; ESP32; machine learning; CNN; LSTM; BiLSTM; GRU.

1. INTRODUCTION

The heartbeat refers to the rhythmic tightening and relaxing of the heart muscles powered by the heart's internal electrical system. This action is essential for pumping blood, which delivers oxygen and nutrients to tissues while removing waste products.

The heart rate (HR) is the number of heartbeats recorded in one minute, typically expressed in beats per minute (BPM). It serves as a critical indicator of how effectively the cardiovascular system is functioning. In healthy adults at rest, the heart rate generally ranges from 60 to 100 BPM, although it can vary based on fitness level, emotional state, and physical exertion. Monitoring heart rate accurately is vital for assessing overall health, detecting medical conditions, and understanding how the body responds to various internal and external factors.

Wearable technology is becoming increasingly essential, especially in areas like fitness and healthcare, where continuous heart rate monitoring plays a critical role. While ECG provides accurate readings, it's impractical for everyday use, leading to the rise of wrist-worn devices using photoplethysmography (PPG), like the Fitbit Charge [2] and Apple Watch

[3]. However, PPG signals are prone to motion artifacts, making heart rate estimation more challenging compared to ECG. Enhancing the reliability of PPG-based predictions remains a key research focus [4].

This paper presents a multi-sensor approach to heart rate estimation by combining data from PPG, GSR, ECG, accelerometer, and temperature sensors. Using machine learning, models are trained on public datasets and validated through a real-time hardware setup powered by an ESP32 microcontroller. The system supports wireless data transmission and real-time analysis, contributing to the development of low-cost, accurate, and scalable solutions for wearable health monitoring.

The system comprises of Heart Rate Monitor sensor AD8232 [6], MAX30102 IR Red LED Photodetector [7], Adafruit-1231 ADXL345 Sensor Image [9], DS18B20 Temperature Sensor Image [10], Tactile Switches [11], and 0.96″ Zoll OLED SSD1306 Display Image [12].

Programming GSR sensors can be learned online [8].

2. SYSTEM DESIGN AND ARCHITECTURE

This system is designed to estimate heart rate by combining wearable sensor data with machine learning models. It is structured around two key phases: the software phase, which focuses on training models using the PPG DaLiA dataset, and the hardware phase, which involves real-time data collection from multiple sensors using an ESP32 microcontroller. Together, these phases enable a practical, real-world solution for accurate and efficient heart rate monitoring [1].

2.1. Software Phase

The software phase of this system focuses on building and validating machine learning models for heart rate estimation using the publicly available PPG DaLiA dataset. To ensure real-world applicability, only data corresponding to sitting and walking activities was selected—two activities feasible for real-time testing in the hardware phase [1].

From the dataset, five sensors were used: ECG, PPG/BVP, accelerometer, EDA, and temperature, chosen based on their relevance to the selected activities. Since the dataset lacked aligned heart rate labels due to different sampling rates, three distinct preprocessing strategies were applied: trimming sensor data to match label count, combining existing and recomputed HR values, and exclusively using HR values recalculated from ECG R-peaks [1].

Neural network models such as CNN, RNN, LSTM, BiLSTM, and GRU were trained using these preprocessed signals to learn patterns linking physiological signals to heart rate. Model performance was evaluated using MAE, MSE, and RMSE, and the best-performing models were saved in .keras format for testing them in the hardware phase. These saved models include the architecture and learned weights but exclude raw training or testing data, preserving only the model's learned behavior [1].

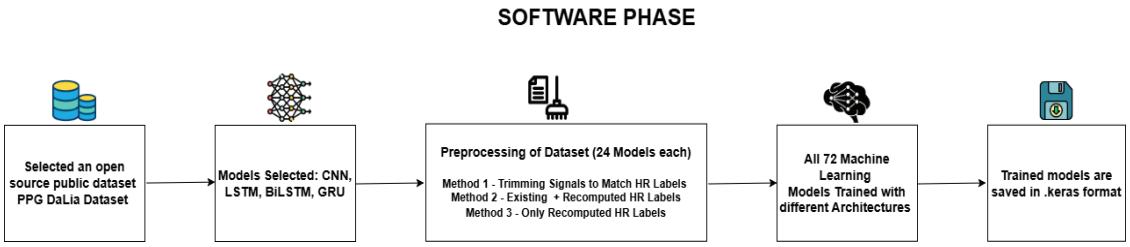

Figure 1. *Software Phase design.*

Three distinct preprocessing methods were applied to align heart rate labels with sensor data: 1. trimming sensor signals to match the existing HR label count, 2. combining existing HR values with those recalculated from ECG R-peaks, and 3. using only recomputed HR labels derived from R-peak intervals. For each method, 24 different models were trained using various combinations of architectures, layers, neuron counts, and hyperparameters, resulting in a total of 72 trained models optimized for heart rate prediction.

The final models were saved in .keras format, preserving the architecture, learned weights, and biases. Notably, the actual training and test datasets were not embedded in the saved files—only the model's learned behaviour was retained. This comprehensive software development phase set the foundation for real-time integration in the hardware phase, enabling scalable, sensor-driven heart rate prediction [1].

2.1.1. Challenges Faced in Software Phase

Several practical challenges were encountered while preparing the PPG DaLiA dataset for training neural networks, particularly around data quality, synchronization, and label alignment. Below are the key issues and how they were resolved:

Challenge 1: Missing Walking Data for Subject S6

Subject S6 had no walking data due to a hardware issue during the original data recording. This was confirmed from the dataset documentation, which stated that only 1.5 hours of data were available for this subject. To prevent runtime errors during training, the code was updated to log a warning and skip missing activity data gracefully [1].

Challenge 2: Mismatch in Sensor Sampling Rates

Each sensor in the dataset had a different sampling rate (e.g., ECG: 700Hz, PPG: 64Hz, EDA & TEMP: 4Hz), leading to a mismatch in data lengths. To resolve this, all five selected sensors (ECG, PPG, ACC, EDA, TEMP) were trimmed to the shortest signal length, ensuring alignment across channels. Only synchronized data related to sitting and walking was retained for training [1].

Challenge 3: Misalignment Between HR Labels and Sensor Data

HR labels were generated independently of sensor streams and had a lower frequency compared to high-frequency sensor data. This caused a significant mismatch between feature and label lengths. To address this, three preprocessing strategies were applied:

Preprocessing Method 1: Trim sensor signals to match HR label count.

Preprocessing Method 2: Combine existing HR labels with those recomputed from ECG R-peaks.

Preprocessing Method 3: Use only recalculated HR labels derived from R-peak intervals.

These approaches ensured proper synchronization between labels and input features, making the dataset suitable for supervised learning [1].

Flow of Software Phase:

Selected Publicly available dataset PPG DaLiA Dataset ➔ Selected CNN, LSTM, BiLSTM, GRU Neural Network Models ➔ Dataset Preprocessing ➔ Model Training ➔ Saved trained models in .keras format.

2.2. Hardware Phase

The hardware phase focuses on building a real-time physiological data acquisition system using a set of wearable sensors interfaced with an ESP32 microcontroller. Acting as the central controller, the ESP32 collects signals from the sensors, performs initial preprocessing, and wirelessly transmits the data for further analysis, thanks to its integrated Wi-Fi capability [1].

The hardware system is composed of the following components:

- ESP32 Microcontroller – core unit for signal collection and wireless communication
- AD8232 ECG Sensor – captures the electrical activity of the heart
- MAX30102 PPG Sensor – used for detecting blood volume changes
- Grove GSR Sensor – measures electrodermal (skin conductance) responses
- ADXL345 Accelerometer – detects movement across three axes
- DS18B20 Temperature Sensor – monitors body temperature
- OLED Display – displays real-time readings or system status
- Tactile Switch – provides manual input control to the system

Together, this setup enables real-time streaming of multi-sensor bio-signals to a connected system for heart rate estimation. The ESP32 acts as the bridge between sensor hardware and software logic, enabling continuous monitoring in a lightweight and portable form.

This section outlines the complete setup of the hardware system, detailing how each sensor is interfaced with the ESP32 microcontroller and how the collected data flows from the sensors to the laptop for further analysis. The architecture is designed to support seamless acquisition of physiological signals in real time, enabling reliable preprocessing and heart rate prediction using trained machine learning models. At the core of the system is the ESP32 microcontroller, which acts as the central hub, connecting and collecting data from five different biosensors. These sensors measure various physiological parameters and transmit their readings to the ESP32. Once received, the data is wirelessly sent to a local computing device, a laptop, for processing and inference. The decision to use a local storage and processing approach was driven by privacy con-

cerns, as data was gathered directly from real human participants. Rather than transmitting sensitive biometric data to the cloud or external servers, all information remained securely on a personal device, reducing risk and having full control over data handling [1].

To ethically manage data collection, all participants were provided with a clear and transparent consent form, outlining what data would be collected, how it would be used, and the steps taken to ensure their privacy. Only after obtaining informed consent was any data recorded [1].

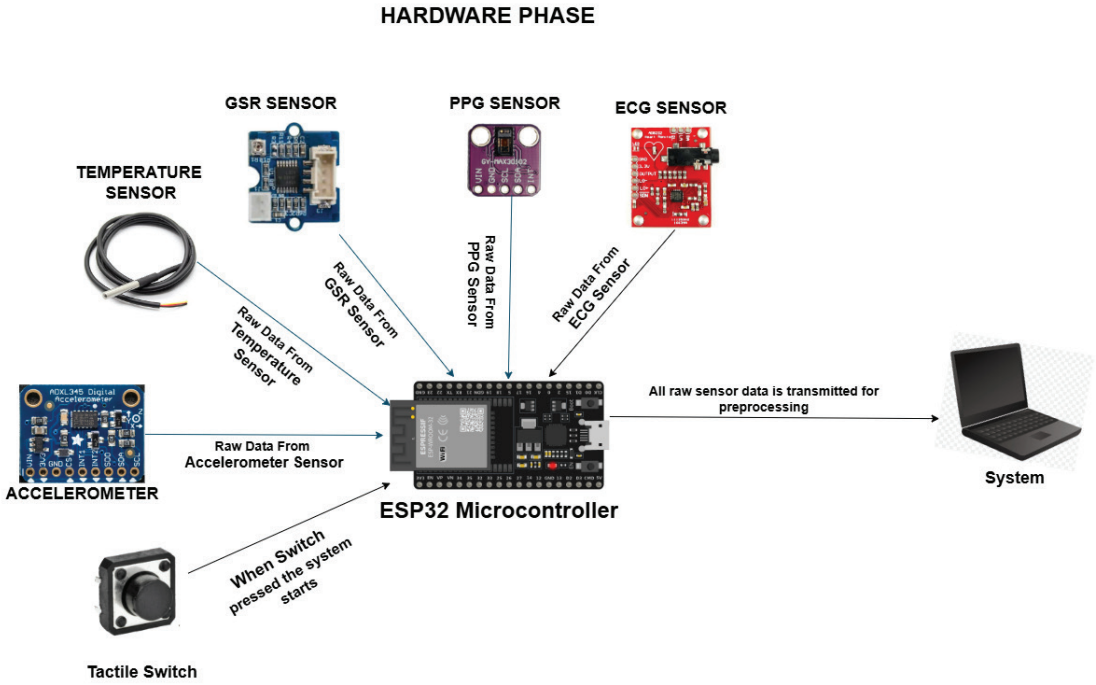

Figure 2. *Architecture Overview of Hardware Phase [5], [6], [7], [8], [9], [10], and [11].*

From the system diagram, it is evident that the ESP32 microcontroller functions as the main unit, interfacing with multiple sensors, an OLED display, and a computer. Once powered on, the ESP32 initializes communication with each connected device, verifying their readiness through a self-check routine. The firmware, developed using the Arduino IDE, is tailored to manage each sensor's specific configuration, such as sampling rate, data acquisition protocol, and power usage. After confirming that all components are properly connected, the system displays the message: "System is Ready, Press Button to Start."

When the tactile switch is pressed, the ESP32 waits for a Wi-Fi connection from the laptop, which acts as the data receiver. Upon successful connection, the microcontroller begins streaming real-time data from the five sensors: MAX30102 (PPG), AD8232 (ECG), DS18B20 (temperature), Grove GSR (EDA), and a 3-axis accelerometer. This raw data is continuously collected over a fixed duration (typically 5–10 minutes) and wirelessly transmitted to the laptop for further processing [1].

2.2.1. Communication Protocols in the Hardware Phase

The sensors used in this hardware setup operate using a mix of communication protocols. While most sensors have built-in analog-to-digital converters (ADCs), two—ECG (AD8232) and GSR (Grove EDA)—output analog signals, which are digitized by the ESP32's internal ADC before being transmitted to a laptop. Regardless of their original format, all sensor data are ultimately transmitted as digital signals, ensuring compatibility with machine learning processing on the receiving system [1].

Three key communication methods were used. I2C (Inter-Integrated Circuit) was employed for digital sensors like the MAX30102, OLED display, and ADXL345. This protocol supports multiple devices using just two lines (SDA and SCL), making it ideal for compact embedded systems. Analog communication was used for ECG and GSR sensors, where raw voltage signals are continuously sampled and converted into digital values. One-Wire communication, used for the DS18B20 temperature sensor, enables data transmission over a single line, simplifying wiring and allowing multiple devices to coexist on the same bus. Each protocol was selected based on the sensor's design and use case, balancing accuracy, power efficiency, and system simplicity [1].

2.2.2. Data Collection Protocol in Hardware Phase

A unique subject ID between 0 and 9 was given to each of the nine subjects, or participants, both male and female, from whom data were gathered during the hardware phase. As was covered in the previous section, the purpose of this data collection was to test and assess the performance of the software phase's pre-trained models using real-time sensor data [1].

As part of the data collection procedure, physiological signals were recorded from subjects performing two specific activities. During the *sitting* activity, subjects remained still in front of the hardware setup while data was continuously collected for 10 minutes. For the *walking* activity, participants walked at a comfortable pace for 15 to 20 minutes before the actual data recording began. To ensure accuracy and reliability, data were collected on multiple occasions for each participant. The overall flow of the hardware phase involved capturing raw data from five different sensors, which was transmitted via the ESP32 microcontroller to a laptop for further processing.

2.3. Integration of Software Phase and Hardware Phase

The integration phase serves as the bridge where the independently developed software and hardware phases are brought together to function as a unified system. In the software phase, a total of 72 machine learning models were trained using the PPG DaLiA dataset, each with varying architectures, layers, neuron counts, and hyperparameters. These models were fine-tuned and stored in .keras format for future inference. On the other hand, the hardware phase involved connecting five physiological sensors to an ESP32 microcontroller, which acted as the central data collection unit. The ESP32 gathered real-time data

from the sensors and wirelessly transmitted it to a laptop. This integration phase is where the real-time sensor data from the hardware phase is fed into the pre-trained models from the software phase, enabling heart rate prediction based on live inputs, thus completing the full cycle from data acquisition to AI-driven inference [1].

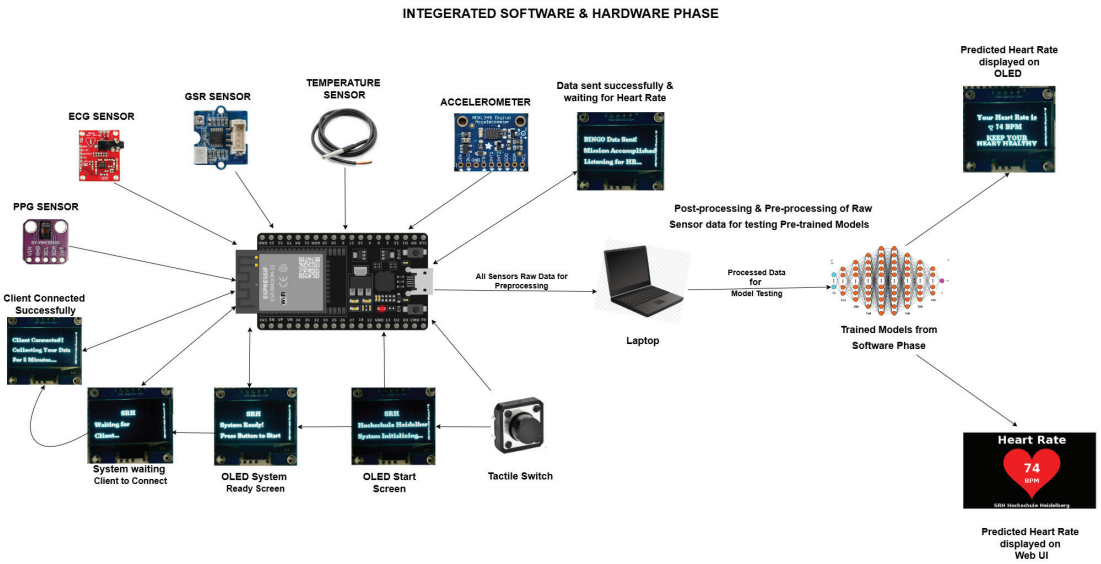

Figure 3. *Architecture of the Heart Rate Estimation System [5], [6], [7], [8], [9], [10], [11], and [12].*

The diagram above presents the complete system architecture of this study as a spectacle of how unseen sensor data collected during the hardware phase is fed into pre-trained .keras models, which were trained on the PPG DaLiA dataset during the software phase, to predict heart rate.

This section outlines the full pipeline for processing real-time physiological signals collected during the hardware phase and preparing them for testing against the pre-trained models developed in the software phase. The dual-phase process includes post-processing the raw sensor data for consistency and noise removal, followed by pre-processing to align with model input requirements [1].

Once data is transmitted from the ESP32 microcontroller to the laptop via Wi-Fi, a custom Python script stores it in .csv and .pkl formats. This raw data contains sensor signals from ECG, BVP, EDA, temperature, and accelerometer sensors, each with distinct sampling rates and potential noise due to motion or environmental factors.

1) Storing Incoming Sensor Data: Incoming raw data is saved as raw_sensor_data.csv, containing timestamps and unfiltered sensor values. This storage preserves the original structure for processing.
2) Signal Cleaning & Preprocessing: To enhance signal quality, various filtering techniques are applied:
   - Bandpass Filtering: ECG (0.5–50 Hz) and BVP (0.5–15 Hz) are filtered to retain heart-related frequencies. EDA and temperature signals are smoothed to remove spikes.
   - Notch Filtering: A 50 Hz notch filter is used to remove the powerline noise.

- Z-Score Normalization: All signals are standardized to mean = 0 and std = 1 for balanced model input. The cleaned data is saved as processed_sensor_data.csv.

1) HR Label Generation via Sliding Window: Heart Rate (HR) labels are computed using an 8-second sliding window (with 2-second shifts) by detecting peaks from ECG (R-peaks) or BVP (as a fallback). R-R intervals are calculated, and the formula HR = 60 / interval (sec) is used to generate BPM values. Each HR label represents the average HR within an 8-second window, ensuring smooth transitions.
2) Synchronization & PKL File Creation: Since sensors operate at varying sampling rates (e.g., ECG at 700 Hz, BVP at 64 Hz, EDA/TEMP at 4 Hz), data is synchronized using timestamps. All sensor readings within an 8-second window are assigned the same HR label. This prevents inconsistencies and enables aligned model input [1].

Each .pkl file contains:

- Time-aligned ECG, BVP, EDA, TEMP, ACC signals
- HR labels matched to the corresponding 8-second windows
- This design manages sensor-sensor and sensor-label mismatches effectively and ensures compatibility with model expectations.

1) Model Prediction using Pre-trained Models: The processed .pkl files are organized subject-wise, each containing multi-session recordings. These are loaded and passed to the 72 pre-trained models (CNN, RNN, LSTM, BiLSTM, GRU) saved in .keras format from the software phase. The models take the sensor data and predict HR values in real time.
2) Real-Time Visualization: Predicted HR values are displayed both on the OLED screen connected to the ESP32 and on a Web UI dashboard [1].

Final Flow of the entire system:

Five Sensors raw data ➔ ESP32 ➔ Laptop (Data Preprocessing and .pkl creation) ➔ Loading PKL file to Pre-trained Models of Software Phase ➔ HR Prediction ➔ Results Displayed on Web App UI and OLED.

3. RESULTS

This section highlights the results obtained from the heart rate estimation system, covering both the software and hardware phases. During the software phase, model performance was assessed using standard evaluation metrics—Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE)—to measure the accuracy of predictions on the training dataset. In the hardware phase, real-time sensor data collected from participants was used to test these pre-trained models, and the same metrics were applied to evaluate how well each model performed under real-world conditions [1].

3.1. Results of the Software Phase

A total of 72 models were evaluated across three different preprocessing strategies, with 24 trained models per method. The most accurate and reliable outcomes from these trials, based on key performance metrics, are highlighted below.

CNN Best Performing Trial of Preprocessing Method 2:

Architecture – There is 1 Input Layer; 4 CNN layers with 256, 128, 64, and 32 neurons with ReLU activation function, kernel size 2, 1 Maxpooling Layer, 1 Dense Layer with 100 numbers of neurons with ReLU activation function, 1 Flatten Layer, and 1 Output Layer.

MAE – 10.655859042778024, MSE – 242.41237147029514, RMSE – 15.569597665652608.

LSTM Best Performing Trial of Preprocessing Method 1:

Architecture – There is 1 Input Layer; 3 LSTM Layers with 128, 64, and 32 neurons with a dropout of 0.2, 2 Dense layers with 128 and 64 numbers of neurons with ReLU activation functions and a dropout of 0.2, and 1 Output Layer.

MAE – 8.07427715704232, MSE – 156.9656917719686, RMSE – 12.528594964000098.

Bi-LSTM Best Performing Trial of Preprocessing Method 2:

Architecture – There is 1 Input Layer, 2 BiLSTM layers with 8,4 neurons with a dropout of 0.2, 1 Dense Layer with 16 numbers of neurons with ReLU activation functions with a dropout of 0.2, and 1 Output Layer.

MAE – 7.3934866673903405, MSE – 137.09411541188987, RMSE – 11.708719631620268.

GRU Best Performing Trial of Preprocessing Method 2:

Architecture – There is 1 Input Layer, 4 GRU layers with 256, 128, and 64 neurons with a dropout of 0.5 and 0.3, 2 Dense layers with 128 and 64 numbers of neurons with ReLU activation functions, and 1 Output Layer.

MAE – 8.893359010141099, MSE – 181.59074108877368, RMSE – 13.475560882158995.

Bi-LSTM Best Performing Trial of Preprocessing Method 3:

Architecture – There is 1 Input Layer, 2 BiLSTM layers with 64,32 neurons with a dropout of 0.2, 1 Dense Layer with 50 numbers of neurons with ReLU activation functions, and 1 Output Layer.

MAE – 1.9532807834843728, MSE – 22.381587964758886, RMSE – 4.730918300368216.

3.2. Results of the Hardware Phase

Real-time data from all five sensors was collected across multiple sessions involving nine individual subjects. This data was tested against all 72 trained models—spanning three different preprocessing techniques—with the best-performing results are mentioned below.

CNN Best Performing Trial of Preprocessing Method 2:

Architecture – There is 1 Input Layer; 4 CNN layers with 256, 128, 64, and 32 neurons with ReLU activation function, kernel size 2, 1 Maxpooling Layer, 1 Dense Layer with 100 numbers of neurons with ReLU activation function, 1 Flatten Layer, and 1 Output Layer.

MAE – 41.149836527493214, MSE – 1886.438650208886, RMSE – 43.43315151136152.

LSTM Best Performing Trial of Preprocessing Method 1:

Architecture – There is 1 Input Layer; 3 LSTM Layers with 128, 64, and 32 neurons with a dropout of 0.2, 2 Dense layers with 128 and 64 numbers of neurons with ReLU activation functions and a dropout of 0.2, and 1 Output Layer.

MAE – 24.73506383666422, MSE – 733.1327101455512, RMSE – 27.076423510972624.

Bi-LSTM Best Performing Trial of Preprocessing Method 3:

Architecture – There is 1 Input Layer, 2 BiLSTM layers with 64 and 32 neurons with a dropout of 0.2, 1 Dense Layer with 50 numbers of neurons with ReLU activation functions, and 1 Output Layer.

MAE – 25.93604230729582, MSE – 813.2612413361254, RMSE – 28.517735557651232.

4. CONCLUSION

This thorough study of heart rate estimation using wearable sensors and machine learning models integrates hardware and software elements for physiological monitoring in real time. The research successfully developed a dependable system that can collect physiological data, preprocess data, train deep learning models, and conduct real-time system testing. This research aimed to bridge the gap between collecting unprocessed sensor data and reliably estimating heart rate for practical applications by leveraging embedded technologies and machine learning [1].

In the software phase, deep learning models such as CNN, LSTM, BiLSTM, and GRU were trained. A total of 72 distinct model trials were carried out to assess different architectures and hyperparameter setups. The models that performed best were selected using their Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).

Following a successful software implementation, the hardware phase was used to evaluate real-time testing. Real-time physiological data was wirelessly sent to a PC for further

processing using an ESP32 microcontroller that was interfaced with the worn sensors. The trained models were then applied to the sensor data in order to estimate heart rate.

In summary, this work combines machine learning and embedded technologies to offer a strong basis for AI-driven health monitoring systems. The effective deployment of this system demonstrates its potential to enhance wearable healthcare devices by guaranteeing consistent, dependable, and real-time heart rate monitoring in real-world scenarios [1].

5. FUTURE SCOPE

In the future, this system can be extended to support a broader range of human activities beyond just sitting and walking. One major improvement would be miniaturizing the hardware into a compact, wrist-worn form factor, making it more practical and user-friendly for daily wear. Furthermore, by incorporating extra sensors to capture additional health indicators such as respiration rate, SpO2, and other vital signs, the system can evolve into a complete health monitoring solution.

Expanding the range of physiological inputs not only improves model accuracy but also opens doors to predicting a wider set of health-related metrics, such as cardiovascular health, oxygen saturation, and sleep quality, through advanced AI models. This creates a more intelligent and responsive system for supporting overall well-being.

In addition, the data collected from the wearable device could be securely uploaded to cloud storage in an encrypted format. Each user would have personalized access to their own health dashboard via a secure login, ensuring that the data remains private while still being readily available to the individual when needed.

FUNDING:
This research received no external funding.

ACKNOWLEDGMENTS:
We would like to thank all the participants for participating in this study's data collection process.

INSTITUTIONAL REVIEW BOARD STATEMENT:
Not applicable.

INFORMED CONSENT STATEMENT:
Informed consent was obtained from all subjects involved in the study.

CONFLICTS OF INTEREST:
The authors declare no conflict of interest.

REFERENCES

[1] Nikhil Kumar, Heart Rate Estimation Using Wearable Sensors and Machine Learning, Master's thesis, Dept. of Information and Technology, SRH Hochschule Heidelberg, Heidelberg, Germany, 2025.

[2] Fitbit. Fitbit Charge 3, 2018. Available online: https://www.fitbit.com/charge3

[3] Apple. Apple Watch Series Official Website. 2019. Available online: https://www.apple.com/lae/watch/

[4] A. Reiss, I. Indlekofer, P. Schmidt, and R. Stiefelhagen, "Deep PPG: Large-Scale

Heart Rate Estimation with Convolutional Neural Networks," Sensors, vol. 19, no. 14, p. 3079, Jul. 2019. Available: https://doi.org/10.3390/s19143079

[5] DOIT ESP32 Devkit V1 Wi-Fi Development Board – Top Straight View, CircuitState, Sep. 2023. [Online]. Available: https://www.circuitstate.com/wp-content/uploads/2023/09/DOIT-ESP32-Devkit-V1-WiFi-Development-Board-Top-Straight-View-CIRCUITSTATE-Electronics-1.jpg

[6] SparkFun Electronics, Heart Rate Monitor sensor AD8232, SparkFun, Available: https://cdn.sparkfun.com/assets/learn_tutorials/2/5/0/HeartRateBoardFront.jpg

[7] MAX30102 IR Red LED Photodetector, How2Electronics. [Online]. Available: https://how2electronics.com/wp-content/uploads/2024/02/MAX30102-IR-Red-LED-photodetector.jpg

[8] All you need to know about GSR sensor – Galvanic Skin Response guides and projects, Seeed Studio[Online]. Available: https://www.seeedstudio.com/blog/2022/03/07/all-you-need-to-know-about-gsr-sensor-galvanic-skin-response-guides-and-projects/

[9] Adafruit-1231 ADXL345 Sensor Image, Distrelec, 2025. [Online]. Available: https://media.distrelec.com/Web/WebShopImages/landscape_large/9-/01/Adafruit-1231-30139029-01.jpg

[10] DS18B20 Temperature Sensor Image, Elektro Turanis, 2025. [Online]. Available: https://elektro.turanis.de/html/prj054/ds18b20.jpg

[11] "Tactile Switches 101," SameSky Devices, 2025. [Online]. Available: https://www.sameskydevices.com/blog/tactile-switches-101

[12] "0.96 Zoll OLED SSD1306 Display Image," Shopify, 2025. [Online]. Available: https://cdn.shopify.com/s/files/1/1509/1638/products/096-zoll-oled-ssd1306-display-i2c-128-x-64-pixel-kompatibel-mit-arduino-und-raspberry-pi-562312.jpg?v=1679397959

Dataset Reference:

"PPG DaLiA Dataset," Kaggle, 2025. [Online]. Available: https://www.kaggle.com/datasets/ameersifat53/ppg-dalia-dataset

Journal of COMPUTER AND FORENSIC SCIENCES

CFS 2025, Vol. 4, Issue 1, pp. 32-61
https://doi.org/10.5937/jcfs4-59602
Original research paper

Application of Time Series Algorithms in Cybersecurity

Marko M. Živanović[1*], Marjan D. Milošević[2], and Emilija Kisić[3]

[1] University Metropolitan, Faculty of Information Technology, Belgrade, Serbia, marko.zivanovic@metropolitan.ac.rs, 0009-0005-9264-3314

[2]University of Kragujevac, Faculty of Technical Sciences, Čačak, Serbia, marjan.milosevic@ftn.kg.ac.rs, 0000-0003-4730-1292

[3]University Metropolitan, Faculty of Information Technology, Belgrade, Serbia,emilija.kisic@metropolitan.ac.rs, 0000-0003-3059-2353

* Corresponding author: marko.zivanovic@metropolitan.ac.rs

Received: June 19, 2025 • Accepted: July 23, 2025 • Published: September 18, 2025

Abstract: This paper explores the application of time series algorithms to enhance anomaly detection in cybersecurity. Windows log files such as PowerShell Operational, Windows Defender, Firewall, System, and others were analyzed, focusing on those with the highest informational potential and data volume. Various models were used: exponential smoothing (Holt-Winters), Prophet, Fourier analysis, and Kalman filter for modeling seasonal, periodic, and linear patterns in system events. Advanced methods include LSTM and GRU neural networks, as well as ensemble algorithms like Random Forest and XGBoost, which demonstrated high accuracy in detecting unusual behavior. Special emphasis was placed on dynamic models, such as Bayesian Structural Time Series, to understand system states over time. Experiments show that applying multiple models enables a robust and adaptive approach to log analysis, especially for early detection of attacks and deviations from norms. The proposed framework highlights the importance of predictive analytics in preventive cybersecurity and provides a foundation for developing intelligent systems for real-time monitoring and response.

Keywords: time series, cybersecurity, LSTM, anomaly detection, XGBoost.

1. INTRODUCTION

In today's rapidly evolving digital landscape, cybersecurity has become a critical concern for organizations and individuals alike. With the increasing complexity and volume of cyber threats, traditional security measures are often insufficient for early detection and prevention of attacks. One promising approach to enhance cybersecurity is the use of time series analysis techniques applied to system log data. System logs, such as those generated by Windows operating systems—including PowerShell Operational, Windows Defender, Firewall, and System logs—contain rich temporal information that can be leveraged to identify abnormal patterns indicative of malicious activities or system faults. Time series algorithms offer powerful tools for modeling and predicting system behavior over time.

Citation: M. Živanović, M. Milošević, and E. Kisić, "Comparative Analysis of Machine Learning Models for Real-time Object Detection," *Journal of Computer and Forensic Sciences*, vol. 4, no. 1, pp. , pp. 32-61 2025, https://doi.org/10.5937/jcfs4-59602.

By capturing seasonal trends, periodic fluctuations, and unexpected anomalies in log data, these methods enable proactive threat detection and improved situational awareness. This paper explores a diverse set of time series techniques ranging from classical statistical models such as exponential smoothing (Holt-Winters), Fourier analysis, and Kalman filters to advanced machine learning approaches including Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) neural networks [1, 2, 3]. Additionally, ensemble learning methods like Random Forest and XGBoost are incorporated due to their proven effectiveness in handling non-linear and complex temporal dependencies [4]. Beyond static models, dynamic frameworks such as Bayesian Structural Time Series [5] and Markov Chains [6] are examined to capture evolving system states and temporal dependencies more effectively. The integration of these multiple modeling approaches aims to provide a robust and adaptive framework for analyzing log data [7], facilitating early detection of cyber attacks and deviations from normal system behavior. The significance of this research lies in its potential to advance predictive analytics in cybersecurity, enabling the development of intelligent monitoring systems that can operate in real-time, reduce false alarms, and enhance response strategies. By harnessing the temporal structure inherent in system logs, this study contributes to building more resilient defenses against increasingly sophisticated cyber threats. The remainder of this paper is organized as follows: *Materials and Methods* describes the data collection process, log sources, and statistical techniques used for analysis; *Results* presents the findings from applying time series models to Windows event logs; *Discussion* evaluates model performance and implications for cybersecurity; and *Conclusion* summarizes key insights and suggests future research directions.

2. MATERIALS AND METHODS

The log files used in this analysis were collected from a local Windows 10 Pro machine using a custom PowerShell script that leverages the **Get-WinEvent** cmdlet. The script was scheduled to run at fixed intervals, every 15 minutes, using the Task Scheduler. Data acquisition was conducted over a continuous period of 60 days, during which the system was used for typical daily activities, including web browsing, software development, system updates, and administrative tasks. This process allowed for the capture of both normal operating conditions and occasional anomalies. The resulting dataset encompasses 12 distinct log sources, with a combined total of 12,046 individual log entries exported in **.csv** format for further preprocessing and time series transformation.

The selected logs span a broad spectrum of system, security, and network events, providing a holistic view of the machine's activity. The general **Application** and **System** logs, each contributing 1,000 informational events, served as a baseline for standard operations and system health. For security-specific insights, several critical logs were included. The **Microsoft-Windows-PowerShell/Operational** log was the most voluminous with 4,007 entries, capturing detailed script execution activities and primarily consisting of "Warning" level events (e.g., Event ID 4104), making it a key source for analyzing command-line behavior. Network activity was monitored through the **Microsoft-Windows-Firewall**

with **Advanced Security/Firewall log,** which recorded 988 "Information" events related to enforced traffic rules, predominantly Event ID 2097 indicating allowed connections.

Further granularity was achieved by incorporating logs that track specific system functions. The **Microsoft-Windows-Windows Defender/Operational** log provided 1,000 informational events detailing routine antivirus scans and updates (most commonly Event ID 1151). User session activities were tracked via the **Microsoft-Windows-Winlogon/Operational** log (1,000 events), while the **Microsoft-Windows-NetworkProfile/Operational** log (1,000 events) recorded changes in network connectivity status (e.g., Event ID 4004). Logs with fewer but highly specific events, such as **Microsoft-Windows-DeviceSetupManager/Operational** (102 events) and **Microsoft-Windows-User Device Registration/Admin** (104 events), offered context on hardware changes and device enrollment. The low event count in the **Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Operational** log (16 events) suggested that remote access was infrequent during the observation period.

The number of recent events exported from each log file for analysis is detailed in Table 1. For each exported event, the following attributes were included: **TimeCreated** (the timestamp of the event), **Id** (the event identifier), **LevelDisplayName** (the severity level, e.g., "Information" or "Warning"), and **Message** (the event description, when available). The complete dataset used in this study is not publicly released due to privacy and security considerations. However, it is available from the corresponding author upon reasonable request for research purposes, under appropriate ethical guidelines.

Table 1. *Log sources used in predictive modeling of Windows events.*

Log Name	Description	Significance for Analysis
Security	Records security-related events such as login attempts, privilege changes, and resource access.	Crucial for detecting unauthorized access attempts and monitoring security incidents.
System	Contains information about system components and drivers, including errors and warnings.	Helps identify issues with hardware and core system services.
Application	Logs events generated by applications installed on the system.	Useful for tracking application performance and errors.
Windows Firewall with Advanced Security	Tracks firewall activity, including blocked and allowed connections.	Important for analyzing network security and identifying potential threats.
DNS Client Operational	Logs DNS client activity, such as queries and responses.	Helps diagnose name resolution and network connection issues.
DHCP Client Operational	Monitors DHCP client activity, including IP address assignments.	Useful for resolving network configuration and connectivity issues.
Sysmon Operational	Records detailed system activity, including process creation and network connections.	Enables advanced threat detection and forensic analysis.
PowerShell Operational	Logs PowerShell usage, including script and command execution.	Important for detecting potentially malicious scripts and automated attacks.
Windows Defender Operational	Tracks Windows Defender activity, including scans and threat detections.	Essential for monitoring antivirus protection and identifying malware.

Terminal Services Local Session Manager	Logs events related to Remote Desktop sessions.	Useful for monitoring access to remote sessions and unauthorized attempts.
Windows Update Client Operational	Tracks activity of the Windows Update client, including installations and errors.	Helps identify issues with system updates.
Task Scheduler Operational	Records scheduled task executions and their statuses.	Important for tracking automated processes and potentially malicious tasks.
Kernel-Power	Logs system power events, such as unexpected shutdowns.	Useful for diagnosing power and system stability issues.
Winlogon Operational	Tracks user login activity on the system.	Helps monitor user sessions and potential security incidents.
Application Experience – Program Inventory	Logs information about installed applications and their changes.	Useful for software inventory and detecting unauthorized installations.
Certificate Services Client Lifecycle System	Monitors certificate-related activity and renewals.	Important for managing certificate-based security infrastructure.
User Device Registration Admin	Logs user device registration events.	Helps manage devices and user access.
User Device Registration Operational	Tracks operational aspects of user device registration.	Useful for diagnosing device registration issues.
Network Profile Operational	Logs changes in network profiles and connections.	Helps monitor network changes and potential security risks.
Network Connection Operational	Tracks establishment and termination of network connections.	Important for analyzing traffic and detecting unauthorized access.
Network Isolation Operational	Logs network traffic isolation events.	Useful for analyzing network segmentation and security.
Network Sharing Operational	Tracks network resource sharing activity.	Helps identify potentially insecure resource sharing.
Group Policy Operational	Logs application and changes of group policies.	Crucial for managing system configuration and security policies.
Windows Defender Antivirus Operational	Tracks activity of the Windows Defender antivirus engine.	Important for malware detection and response.
Windows Defender ATP Operational	Logs advanced threats detected by ATP.	Enables detection of sophisticated attacks and threats.
User Profiles Service Operational	Monitors loading and management of user profiles.	Helps resolve issues with user profiles and sessions.
Device Setup Manager Operational	Logs device installation and configuration events.	Useful for diagnosing hardware and driver-related issues.
Security-SPP Operational	Monitors software license protection activity.	Important for license management and software protection.
Diagnostics Performance Operational	Logs system performance and potential bottlenecks.	Helps optimize performance and identify issues.
Remote Access Operational	Tracks remote access activity.	Crucial for monitoring VPN connections and remote logins.
Remote Desktop Services – RdpCoreTS Operational	Logs Remote Desktop Protocol (RDP) session activity.	Important for detecting and tracking remote sessions.
Remote Desktop Services – RdpCoreTS Debug	Provides detailed debug information for RDP sessions.	Useful for in-depth troubleshooting of remote desktop issues.

In the scope of this research, special attention is given to a subset of event logs deemed particularly relevant for system security, operational integrity, and anomaly detection. The

selected logs provide a multifaceted view of system activity, user behavior, and security-related events. Below is an overview of the key logs included in the analysis:

• **System –** Contains events related to system-level operations, including driver errors, service startups, and hardware-related notifications. This log is essential for tracking the health and stability of the operating system over time.

• **Application –** Captures events generated by user-installed applications. Analyzing this log helps in identifying software crashes, misconfigurations, or abnormal usage patterns.

• **Security –** One of the most critical logs for forensic and auditing purposes, it includes login attempts, privilege use, and access control events. It is fundamental for tracking unauthorized access attempts and auditing user activity.

• **Microsoft-Windows-PowerShell/Operational** – Provides a record of PowerShell script executions, including commands issued interactively or via scripts. This log is especially important for detecting advanced persistent threats (APTs) and automation misuse, as PowerShell is a common vector in cyberattacks.

• **Microsoft-Windows-Windows Defender/Operational –** Logs antivirus scans, malware detections, and remediation actions taken by Windows Defender. This log is crucial for identifying security threats and understanding how the system reacts to them.

• **Microsoft-Windows-Windows Firewall with Advanced Security/Firewall –** Captures data on allowed and blocked network traffic as determined by firewall rules. It provides insights into external connections and can reveal attempts to breach network defenses.

• **Microsoft-Windows-NetworkProfile/Operational** – Records changes in network connectivity and profile assignments (e.g., private, public). This log is helpful for tracking when and how a machine switches between networks—often an indicator of mobile use or exposure to unknown networks.

• **Microsoft-Windows-DeviceSetupManager/Operational –** Contains entries related to hardware detection and driver installation. This can be used to identify unauthorized device usage or to correlate issues caused by newly introduced hardware.

• **Microsoft-Windows-RemoteDesktopServices-RdpCoreTS/Operational –** Logs activities related to Remote Desktop Protocol (RDP) sessions. It plays a key role in monitoring remote access, which is frequently exploited in targeted attacks and lateral movement.

• **Microsoft-Windows-WindowsUpdateClient/Operational –** Records system update events, including successful or failed updates. This information is useful for ensuring system compliance and can also highlight moments of increased system load or instability.

Together, these logs offer a comprehensive temporal dataset suitable for anomaly detection, time-series modeling, and correlation analysis across security, application, and system layers. Their diversity ensures a holistic view of the environment, enabling deeper insights into both routine operations and irregular behaviors.

For the purposes of this analysis, fundamental descriptive and inferential statistical models were applied to quantify the characteristics of event time series extracted from operating system log files. All logs were analyzed using statistical metrics, including **count**, **mean**, **median**, **standard deviation**, **variance**, **range** (min–max), **quartiles** (Q1, Q3), **interquartile range** (IQR), **coefficient of variation** (CV), **skewness**, **kurtosis**, **first-order autocorrelation**, **linear hourly trend**, and the **most frequent event type** and **ID**.

The arithmetic mean represents the central value of a dataset and is used to describe the average number of events within the observed time intervals.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad , \tag{1}$$

Where x_i is the i-th value in the observed dataset, n is the total number of observations, $\bar{x}$ is the arithmetic mean [10].

Standard deviation quantifies the average deviation of values from the mean and is one of the most important measures of data dispersion.

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \quad , \tag{2}$$

Where $(x_i - \bar{x})^2$ is the squared deviation of each value from the mean, $n-1$ is Bessel's correction for estimating population variance from a sample [11];

Variance is the square of the standard deviation and serves as a measure of total dispersion within the dataset. Variance is particularly useful when comparing variability across different logs, especially in analytical modeling contexts [12].

$$Var(X) = \sigma^2 \tag{3}$$

The coefficient of variation is a relative measure of dispersion and is used to compare variability across datasets with different means. CV is a dimensionless quantity, usually expressed as a percentage. A value of CV > 1 indicates high variability, while CV < 0.5 suggests a stable and predictable pattern [13].

$$CV = \frac{\sigma}{\bar{x}} \tag{4}$$

Skewness measures the degree to which a dataset deviates from a symmetric normal distribution. Skewness is particularly useful for detecting extreme values and anomalies in log data [14].

$$Skew(X) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{\sigma}\right)^3 \quad , \tag{5}$$

Interpretation:

$skew > 0$ — the distribution is right-skewed;

$skew < 0$ — the distribution is left-skewed;

$skew \approx 0$ — approximately symmetric distribution.

Kurtosis quantifies how much of the data is concentrated near the mean compared to a normal distribution. It is used to detect "peakedness" and the presence of heavy tails [15].

$$Kurt(X) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{\sigma}\right)^4, \tag{6}$$

Interpretation:

$kurt > 3$ — leptokurtic distribution (sharp peak, heavy tails);

$kurt < 3$ — platykurtic distribution (flat distribution);

$kurt \approx 3$ — normal distribution.

High kurtosis may indicate rare but intense events in the logs (e.g., bursts of PowerShell executions or spikes in network access).

Autocorrelation measures the correlation between a value and its preceding value in a time series, revealing patterns or periodicity in the data [16].

$$\rho_1 = \frac{\sum_{t=2}^{n}(x_t - \bar{x})(x_{t-1} - \bar{x})}{\sum_{t=2}^{n}(x_t - \bar{x})^2}, \tag{7}$$

Interpretation:

$\rho_1 > 0$ — positive autocorrelation;

$\rho_1 < 0$ — negative autocorrelation;

$\rho_1 \approx 0$ — no temporal dependency detected.

In system logs, autocorrelation can help identify recurring cycles (e.g., scheduled tasks or scanning behavior). To detect trends in event counts over time, linear regression is applied to the number of events per unit of time (e.g., per hour) [17]. The basic regression model is:

$$y = \beta_0 + \beta_1 t + \varepsilon, \tag{8}$$

Where:

y — the number of events;

t — time (in hours);

β_0 — intercept (initial value);

β_1 — slope (rate of change);

ε — error term.

A positive β1 indicates an increasing trend in activity over time, while a negative value suggests a decline.

Quartiles divide a dataset into four equal parts:

Q_1 — first quartile (25th percentile);

^{2} — median (50th percentile);

Q_3 — third quartile (75th percentile).

The interquartile range (IQR) is defined as:

$$IQR = Q3 - Q1, \tag{9}$$

IQR measures the spread of the middle 50% of the data and is commonly used in outlier detection. Values outside the range $[Q_1 - 1.5 * IQR, Q_3 + 1.5 * IQR]$ are considered potential outliers [18].

3. STATISTICAL ANALYSIS OF KEY EVENT METRICS

The analysis of four key event metrics across the log files reveals important patterns in system behavior. First, the standard deviation graph (**Figure 1**) shows how much the number of events varies per hour: PowerShell and Firewall logs exhibit very high variability (94.57 and 70.88), indicating sudden and irregular spikes in activity, whereas Windows Defender shows stability with the lowest deviation (~2.4). Next, the analysis of hourly event trends (**Figure 2**) reveals that PowerShell records a strong negative trend (−12.82), suggesting a decline in activity following an initial "burst" (e.g., script executions), while logs such as System and Application remain mostly stable, without a pronounced trend. The coefficient of variation (std/mean) (**Figure 3**) further confirms instability in Firewall (1.79), PowerShell (1.66), and Application (1.52), while Defender is the most stable (~0.78), indicating consistent behavior over time. Finally, the lag-1 autocorrelation analysis (**Figure 4**) shows that the Winlogon log is strongly temporally correlated (0.426), likely due to cyclical user logins, whereas PowerShell shows a negative correlation (−0.327), a clear indicator of "burst" activity—if one hour was active, the next probably was not. **In conclusion**, these insights are of great importance for anomaly detection and predictive modeling applications, where logs with stable or predictable patterns (such as Winlogon or Defender) may be better suited for classical models like ARIMA/SARIMA. ARIMA (AutoRegressive Integrated Moving Average) and SARIMA (Seasonal ARIMA) are statistical models for time series analysis that combine autoregression, integration (differencing to achieve stationarity), and moving averages, with SARIMA additionally incorporating seasonal components. **On the other hand**, logs with high variability may require more robust or specialized approaches [19, 20, 21].

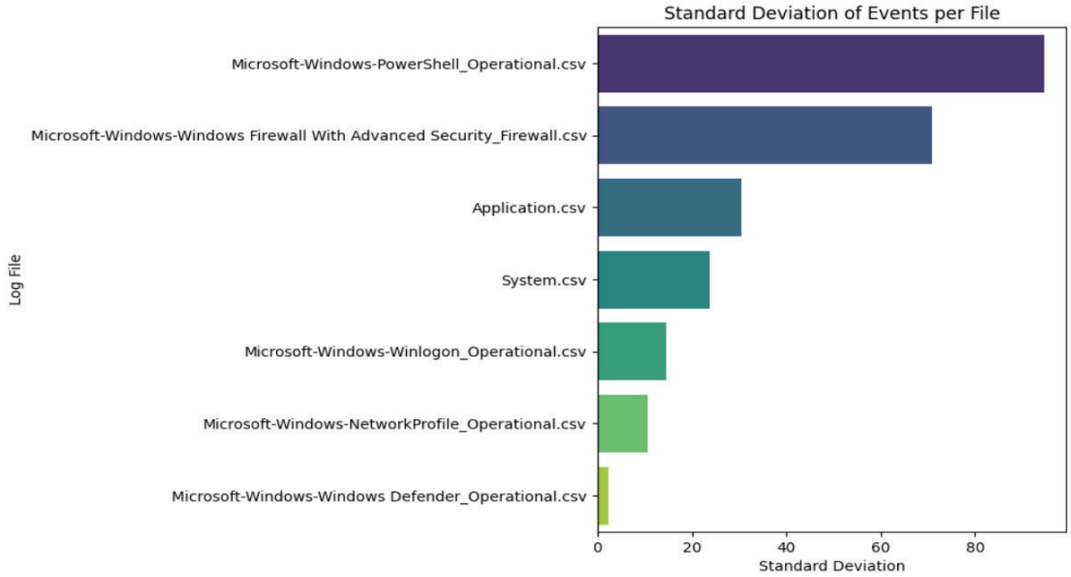

Figure 1. Standard Deviation of Events per File.

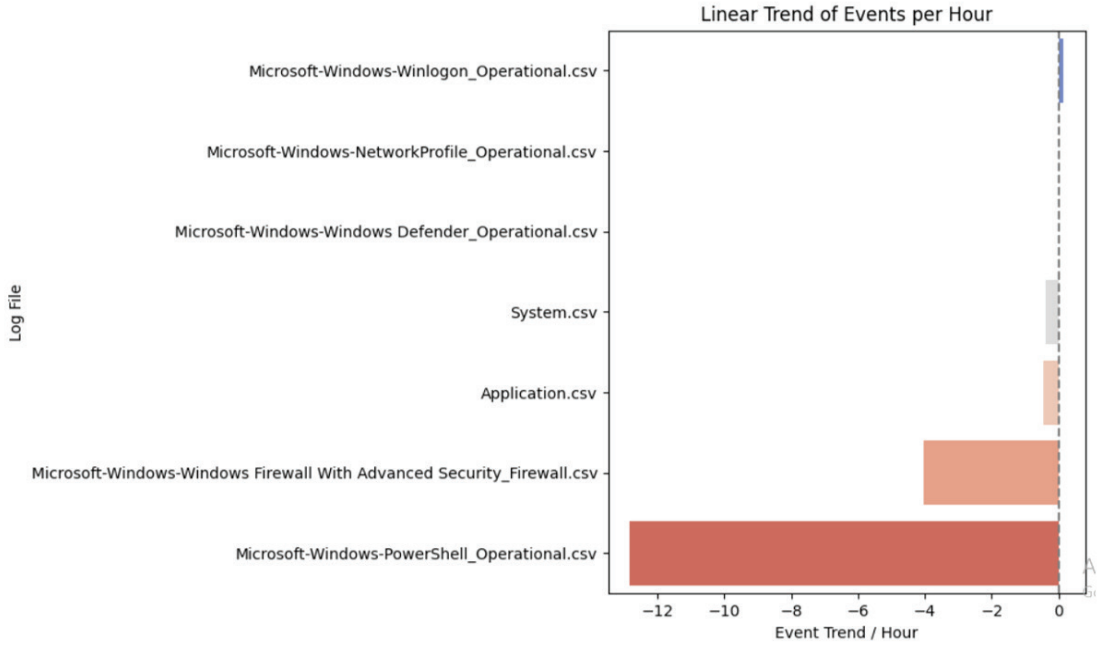

Figure 2. Linear Trends of Events per Hour.

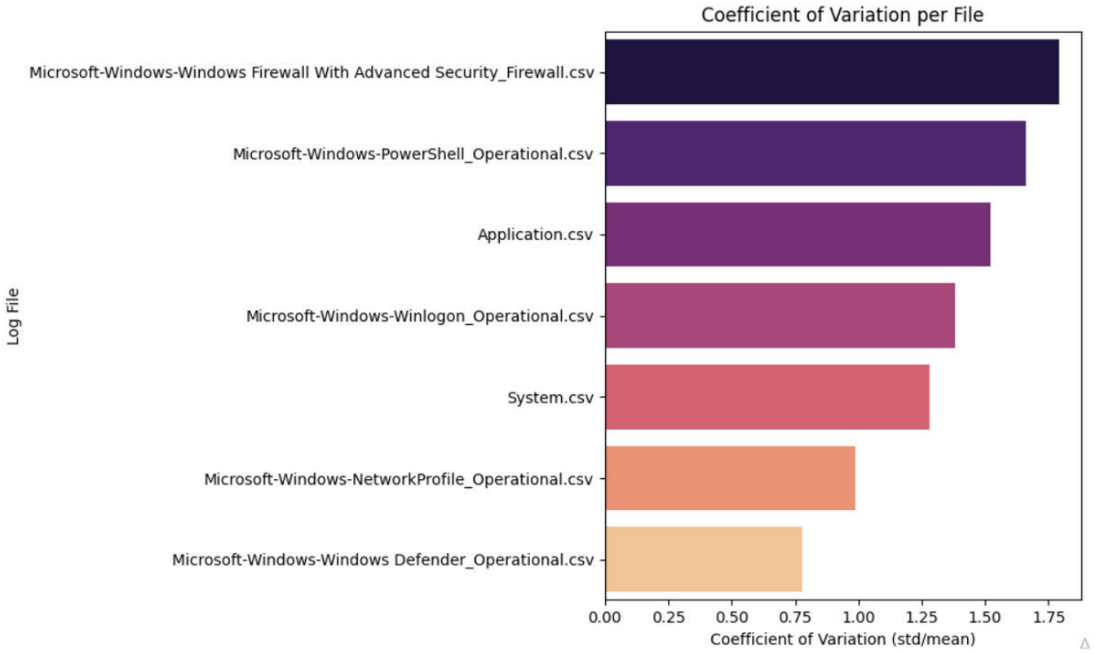

Figure 3. Coefficient of Variation per File.

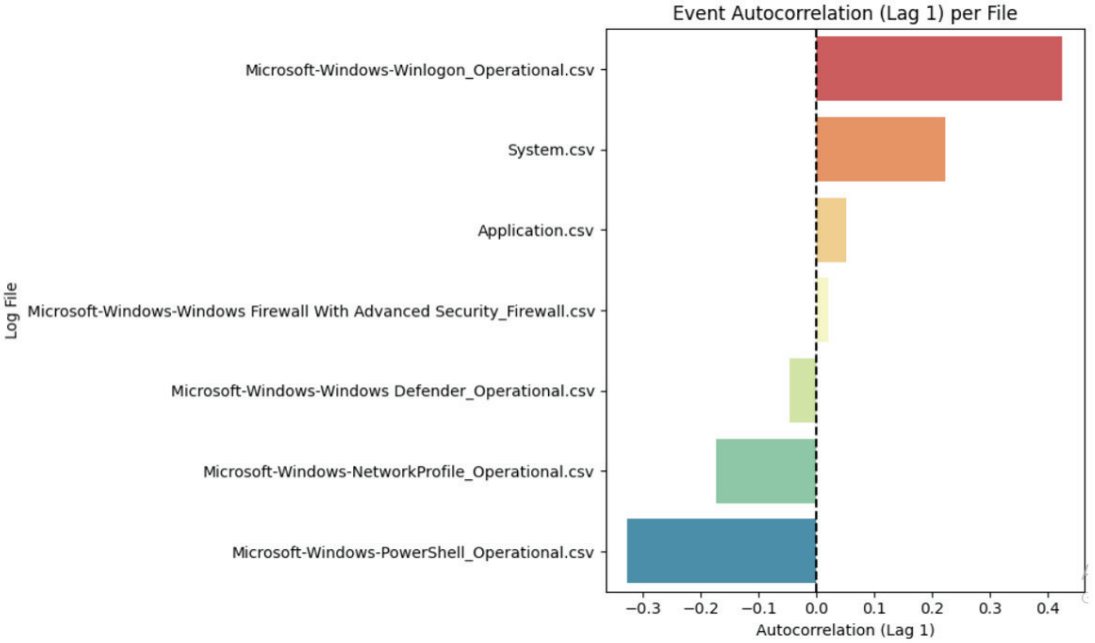

Figure 4. Event Autocorrelation per File.

3.1. Microsoft-Windows-PowerShell/Operational

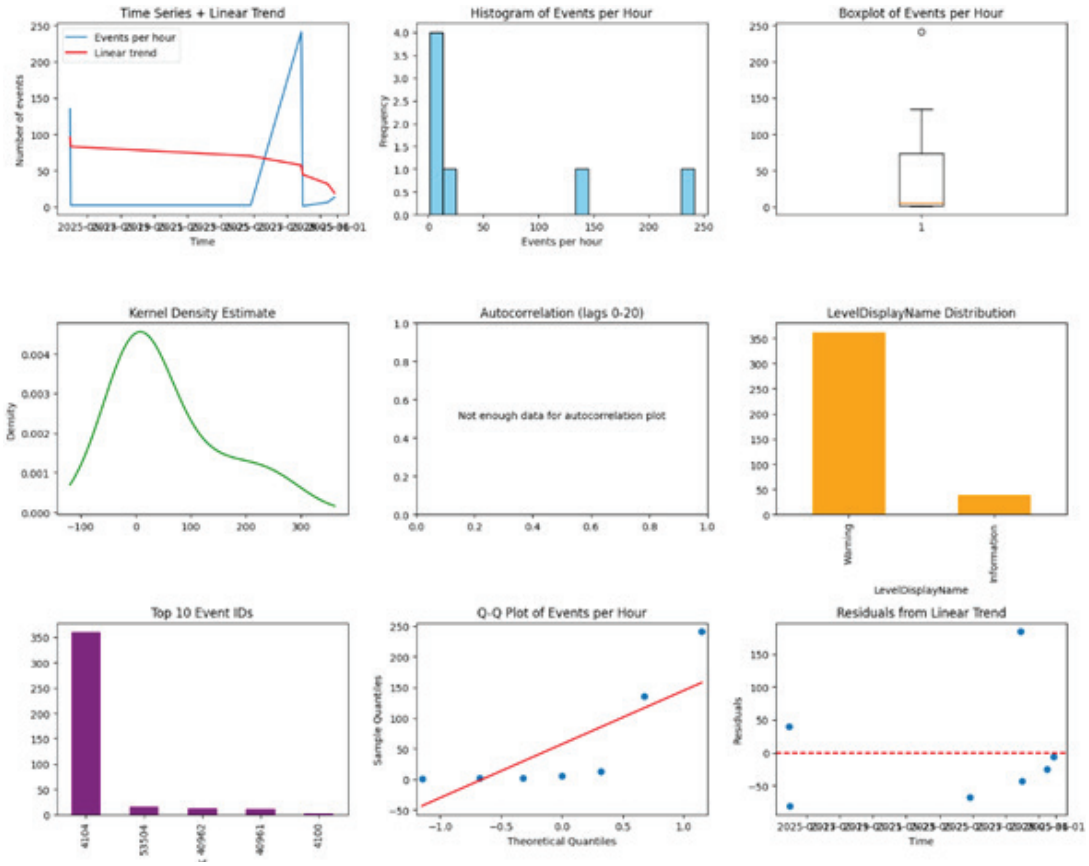

Figure 5. Microsoft-Windows-PowerShell/Operational Statistics.

As presented in Figure 5, analysis of the displayed charts reveals a clear pattern of PowerShell activity that is mostly nonexistent, except for one sudden and highly intense spike. The time series plot clearly illustrates this event through a long period of zero activity, which ends with a sharp jump to over 250 events in a single hour. Although the mathematically calculated trend line shows a decline, it is misleading, as it is heavily influenced by the long period of inactivity rather than the spike itself.

The unusual nature of this event is confirmed through multiple distribution charts; the histogram, boxplot, kernel density plot, and Q-Q plot unanimously indicate that the data drastically deviate from a normal, regular distribution, given that almost all activity is concentrated in a single extreme point (outlier). A key insight into the nature of this event is provided by the identification of the logs themselves: the "Top 10 Event IDs" chart reveals that nearly all activity originated from Event ID 4104, which specifically logs the execution of PowerShell scripts. Moreover, the majority of these events are classified as "Warning," indicating that the system itself flagged this activity as something to pay attention to. The conclusion, therefore, is that the charts do not depict regular activity but rather an isolated case of massive execution of an automated script.

3.2. Microsoft-Windows-Windows Defender/Operational

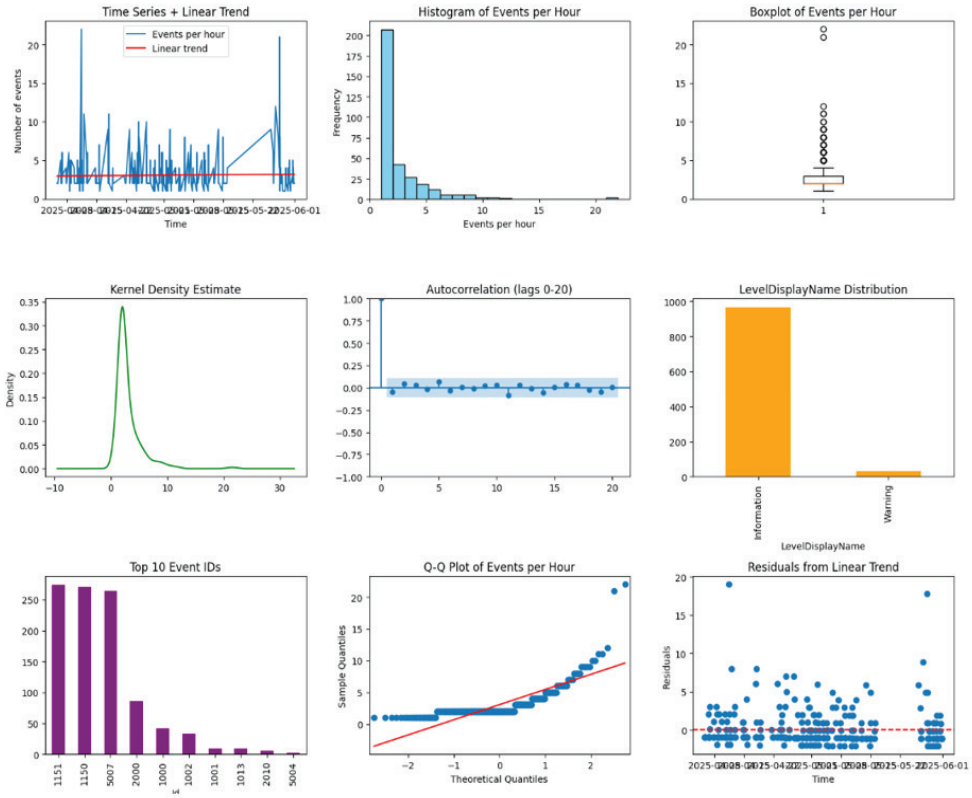

Figure 6. Microsoft-Windows-Windows Defender/Operational Statistics.

As presented in Figure 6, a **comprehensive analysis of the charts shows that the activity of the Defender antivirus program is constant and stable, but with frequent, occasional spikes.** The time series plot displays low but steady activity with regular spikes, while the flat red trend line confirms that there is no long-term increase or decrease in activity. This nature of "constant operation with occasional load" is further supported by the distribution charts; the histogram, boxplot, and kernel density plot clearly show that most events are of low frequency, but with a long "tail" of less frequent yet more intense activity, creating strong positive skewness.

The fact that the data do not follow an ideal "normal" distribution, as seen in the Q-Q plot, is entirely expected for this type of operation. Key context is provided by examining the type and level of events: the "Top 10 Event IDs" chart shows that the activity originates from various operations, with dominant Event IDs 1151 and 1150, which point to routine protection functions.

Most importantly, the "LevelDisplayName Distribution" chart reveals that the vast majority of these events are classified as mere "Information," rather than "Warning." Therefore, these charts do not indicate a problem but rather depict a portrait of a healthy and active antivirus solution performing its regular duties, such as scanning or updating, and generating informational logs in the process.

3.3 Microsoft Windows-Windows Firewall with Advanced Security/Firewall

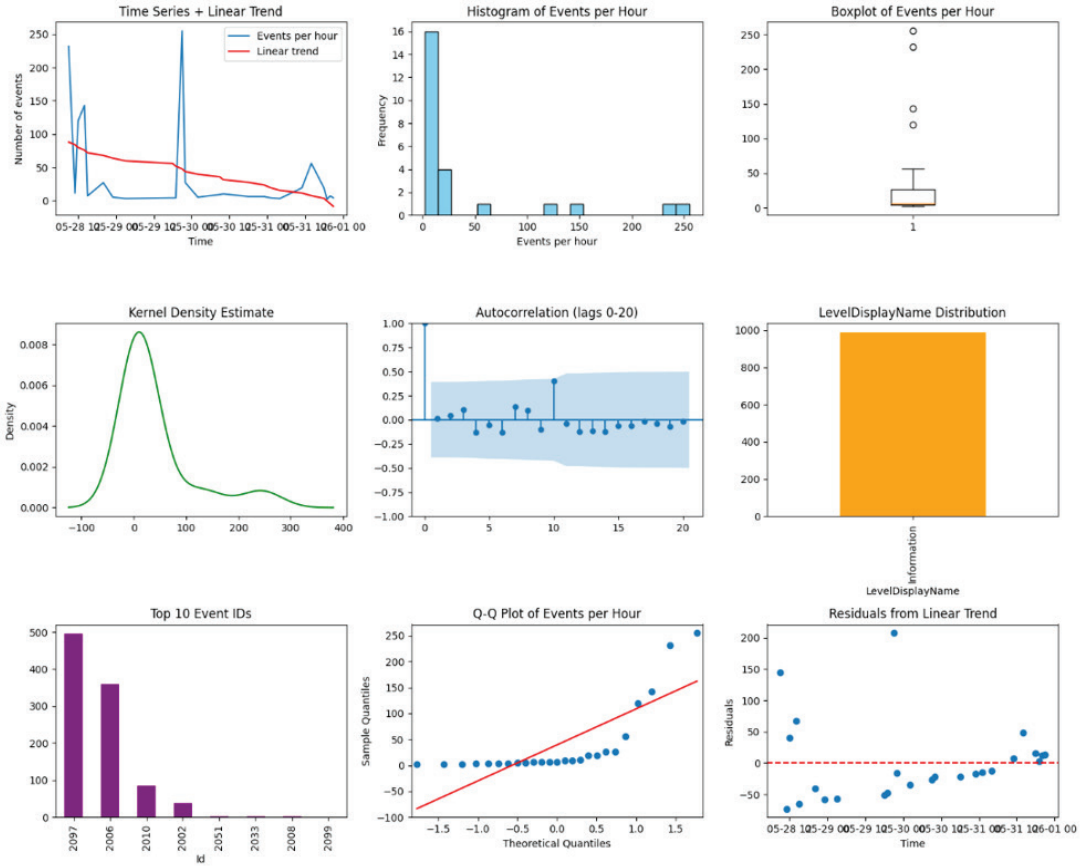

Figure 7. Microsoft-Windows-Windows Firewall with Advanced Security/Firewall Statistics.

The analysis of the Windows Firewall charts reveals activity marked by several pronounced but rare spikes in the number of events, which became less frequent over time. The time series plot clearly illustrates these spikes, as well as a significant downward trend (red line), suggesting that the overall frequency of events decreased during the observed period. The distribution of these events, as shown in the histogram, boxplot, and kernel density plots, confirms that most hours had very little activity, while a few hours experienced an extremely high number of events, resulting in strong positive skewness. However, a key piece of information that alters the interpretation of these "intense events" comes from analyzing their nature. The "LevelDisplayName Distribution" chart unambiguously shows that every single one of these events is classified as "Information," with not a single "Warning" or "Error" present. Furthermore, the "Top 10 Event IDs" chart reveals that the dominant events are ID 2097 and 2006, which typically refer to specific network operations or rule status updates, rather than blocked attacks. The conclusion is therefore more precise: although the firewall recorded occasional "storms" of events, they did not represent security threats. Instead, they were instances of mass generation of informational logs related to specific, likely automated, network processes whose frequency decreased over time.

3.4. Application

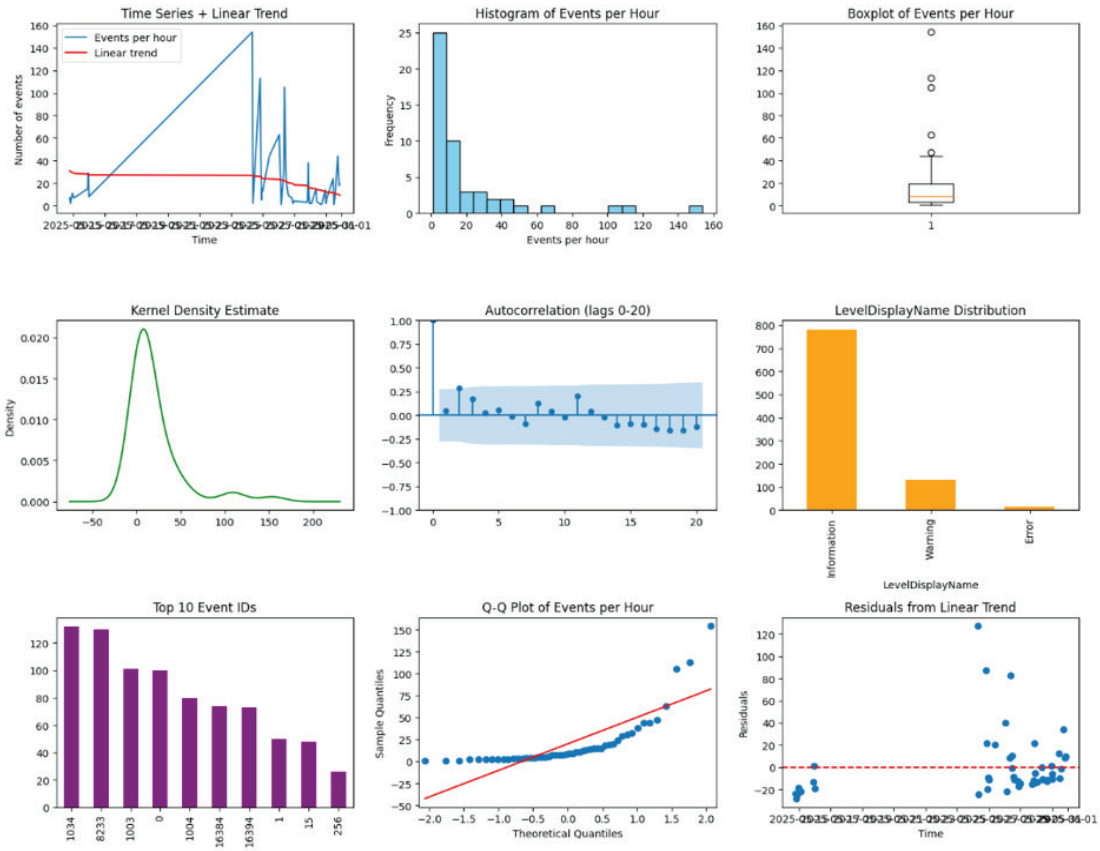

Figure 8. Application Statistics.

An analysis of the application messages and error charts reveals highly unstable activity characterized by occasional but intense spikes in the number of events. Statistically, the data shows a mean of approximately 20 events per hour but with a significant positive skewness of ~2.8 and high kurtosis of ~7.99, indicating that while most hours have low activity, there are frequent and extreme deviations. While the time series and distribution plots (Histogram, Boxplot) visualize this volatile, right-skewed pattern with numerous outliers, the key insight comes from the nature of the events themselves. The "LevelDisplayName Distribution" chart shows that the overwhelming majority of events are informational, not critical errors. However, the "Top 10 Event IDs" (such as 1034, 8233, and 1003) link these informational storms to specific application processes and instability, such as events preceding or following application crashes. Therefore, although the spikes are not composed of critical errors, their origin in software misbehavior makes this log category extremely valuable for diagnosing issues like crash dumps (Figure 8).

3.5. Microsoft Windows Network Profile/Operational

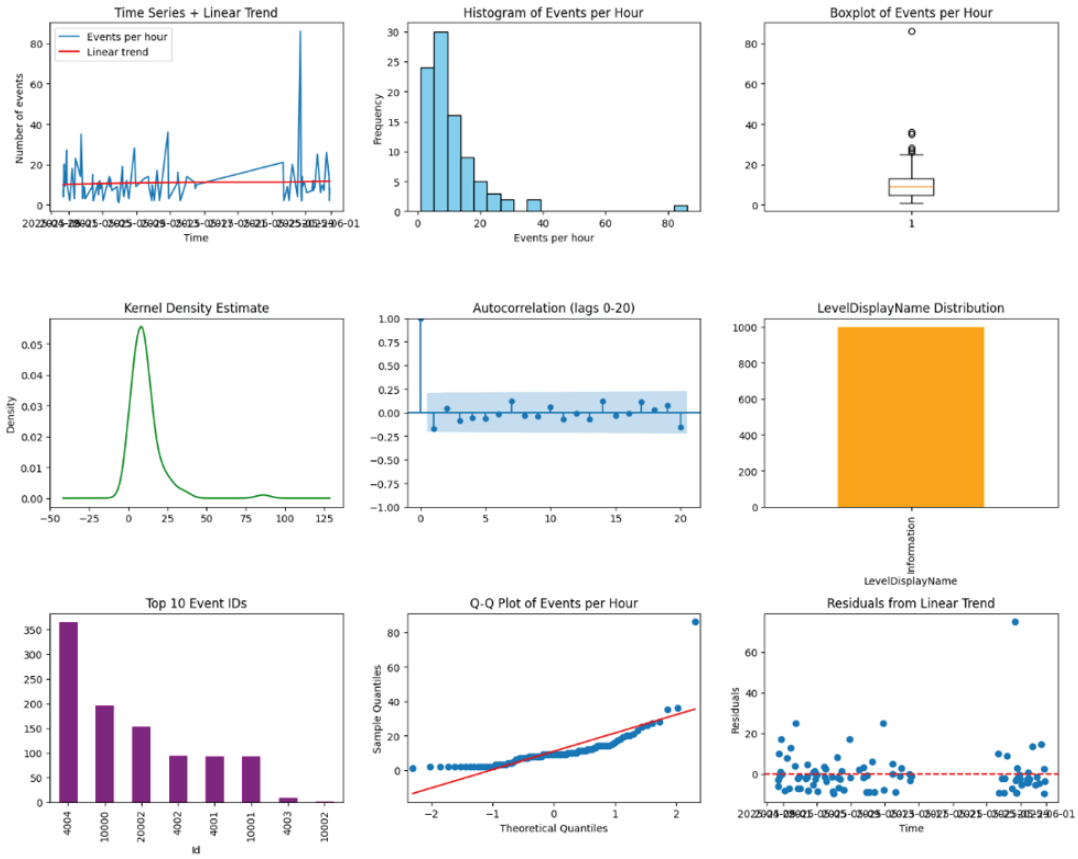

Figure 9. Microsoft-Windows-NetworkProfile/Operational Statistics.

The analysis of the provided charts indicates highly unstable activity, characterized by sporadic but very intense spikes in the number of events. The time series shows periods of low activity interrupted by sudden peaks reaching up to 80 events per hour. The linear trend is nearly flat, suggesting that there is no clear increase or decrease in average activity over the observed period—volatility is the dominant characteristic. The statistical distribution of events per hour, as displayed in the histogram and boxplot, clearly reveals strong right (positive) skewness. Most hours record a very low number of events (with a median below 10), but occasional extremes are evident through numerous outliers. The Q-Q plot further confirms that the data significantly deviate from a normal distribution, which is typical for datasets with rare but extreme events. A key insight is gained through the analysis of the nature of the events themselves. The "LevelDisplayName Distribution" chart unambiguously shows that all recorded events are informational in nature ("Information"), with no critical errors or warnings. Further analysis via the "Top 10 Event IDs" chart reveals that these informational spikes are predominantly driven by specific events, particularly Event ID 4004, which appears significantly more frequently than others. Additionally, the Autocorrelation Function (ACF) plot shows no statistically significant correlation between the number of events in consecutive hours. This suggests that the activity

spikes are random and unpredictable, rather than part of a recurring cycle. In conclusion, although the system does not report any critical errors, it generates occasional "information storms" (log storms) originating from very specific processes (e.g., the one related to ID 4004). While this behavior does not indicate system failure, it may be a symptom of inefficiencies or configuration issues. Monitoring these events is essential for optimizing performance and reducing log "noise," which can help in detecting real problems when they occur (Figure 9).

3.6. System

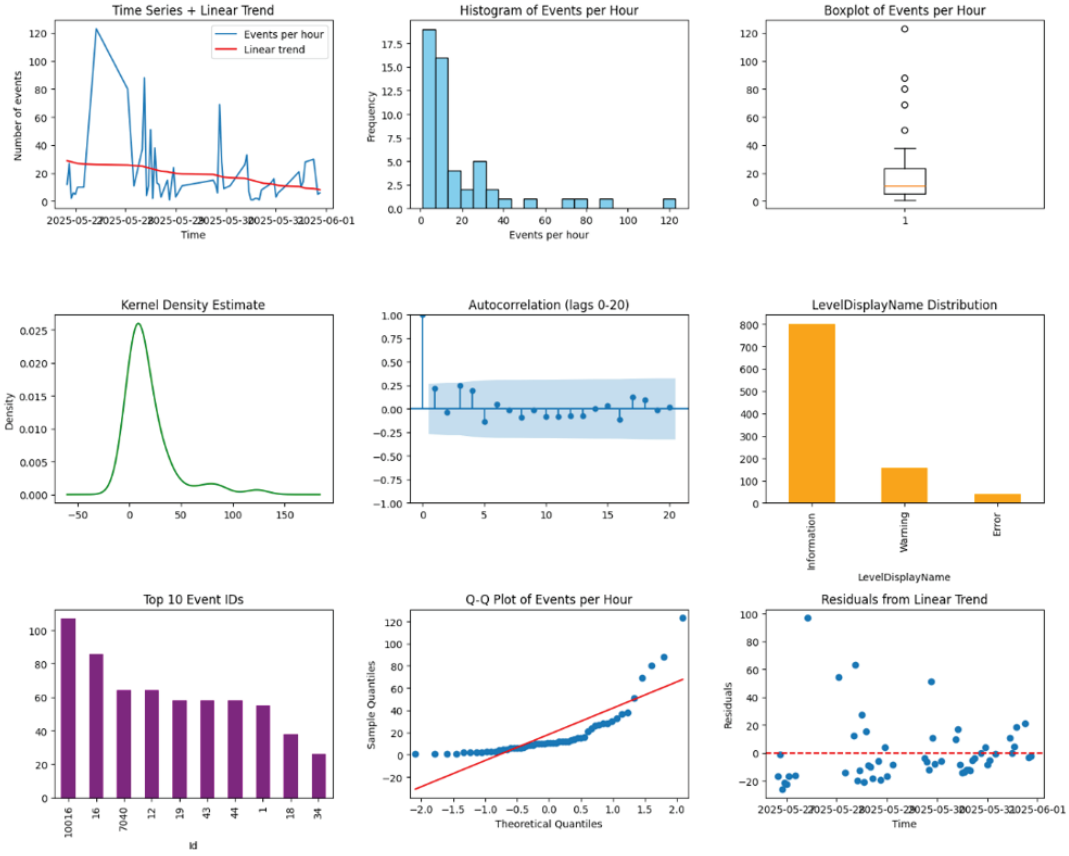

Figure 10. System Statistics.

The analysis of the presented charts reveals a highly variable nature of system events, with one prominent initial spike in activity exceeding 100 events per hour. Unlike the previous example, a clear downward linear trend is observed here, suggesting that the average number of events decreased over the observed period—likely as a result of some intervention or system stabilization. The distribution of events per hour, as shown in the histogram, boxplot, and KDE plot, exhibits pronounced right skewness. Most hours register a low number of events, while rare but intense spikes create a long right tail and numerous outliers. The Q-Q plot confirms that the data do not follow a normal distribution. The key difference

and most important insight lie in the nature of the events themselves. The "LevelDisplayName Distribution" chart shows that, although informational events ("Information") are the most frequent, the system also generates a significant number of warnings ("Warning") and, to a lesser extent, errors ("Error"). This points to actual issues within the application, rather than mere "noise" in the logs. The "Top 10 Event IDs" chart shows that the activity is generated by several different events, with Event ID 10016 being the most frequent. The Autocorrelation Function (ACF) chart, similar to the previous case, shows no temporal dependence, indicating that the occurrence of these events cannot be predicted based on prior activity. In conclusion, the data suggest the presence of instability or issues at the beginning of the observed period, which gradually subsided (as indicated by the downward trend). The presence of warnings and errors, although in the minority, makes these logs particularly important for diagnostics. Analyzing the events behind IDs such as 10016 and 7040—both associated with warnings—is key to understanding the root cause of the problem and preventing its recurrence (Figure 10).

3.7. Winlogon/Operational

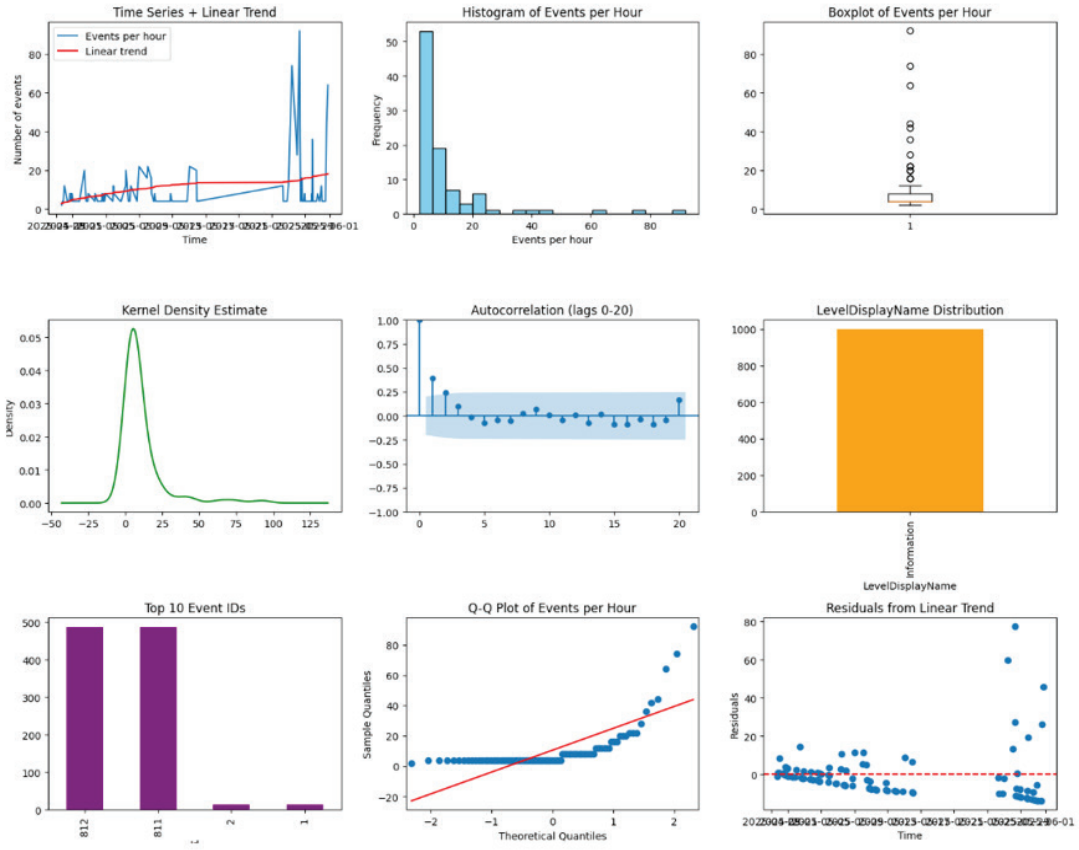

Figure 11. Winlogon/Operational Statistics.

As presented in Figure 11, t**he analysis of the provided charts reveals a pattern of activity characterized by extreme instability and occasional, very strong spikes in the number of events.**The time series shows that, in addition to random peaks exceeding 80 events, there is also a slight upward linear trend. This indicates a gradual increase in the average number of events over the analyzed period.

The statistical distribution is highly right-skewed, as seen in the histogram, boxplot, and KDE plot. The vast majority of hours record a minimal number of events (with a median close to zero), while extreme values occasionally occur, manifested as numerous outliers. The Q-Q plot definitively confirms that the data do not follow a normal distribution due to the heavy right tail. A key insight comes from the analysis of the type and source of events. The "LevelDisplayName Distribution" chart shows that 100% of the events are informational in nature ("Information"). Even more importantly, the "Top 10 Event IDs" chart reveals that nearly all activity is generated by just two IDs: 812 and 811. These two events account for the vast majority of all logs, meaning the "information storms" originate from a highly concentrated source. The Autocorrelation Function (ACF) plot suggests that the activity is mostly random. While there is a borderline significant correlation at the first time lag (lag 1), it is not strong enough to indicate a clear, predictable pattern. Thus, the activity spikes can still be considered unpredictable. In conclusion, the system does not report any errors but suffers from extreme "chattiness" caused by two specific processes (associated with IDs 812 and 811). The upward trend in this activity suggests that the problem may become more pronounced over time. Although not critical, such behavior unnecessarily burdens the logging system and complicates monitoring. Investigating the cause of these messages is recommended in order to reduce informational "noise" and optimize application performance.

3.8. Terminal Services Local Session Manager/Operational

This analysis covers a longer time period (approximately 10 months) and reveals generally low but stable activity. The time series shows that the number of events per hour is mostly in the single digits, with occasional minor spikes reaching a maximum of 35. The linear trend is completely flat, confirming that there is no long-term increase or decrease in activity; the system is in a stable, though occasionally "noisy," state. All distribution charts (Histogram, Boxplot, KDE) indicate extreme right skewness. The vast majority of data is clustered around zero, meaning that there are often hours with no events at all. Rare events appear as outliers. The Q-Q plot further confirms a significant deviation from a normal distribution. Key insights come from the analysis of the nature and temporal structure of the events: the **event type** shown in the "LevelDisplayName Distribution" chart indicates that the events are predominantly informational, but it is important to note that errors ("Error"), although very rare, are also recorded and require attention. The **source of the events**, according to the "Top 10 Event IDs" chart, shows that Event ID 40 is the most frequent trigger of activity, followed by several others with lower frequency. As for **temporal patterns**, unlike previous analyses, the Autocorrelation Function (ACF) chart here reveals statistically significant correlation at multiple time lags (e.g., lags 1, 16, 18), sug-

gesting that the events are not entirely random and that a subtle periodic process may be driving them. In conclusion, the system is stable over the long term and shows no signs of escalating problems. However, it is not without flaws. The presence of even rare errors calls for further investigation. The significant autocorrelation suggests the existence of hidden, periodic patterns in system behavior. It is recommended to examine the processes related to the most frequent event IDs (especially ID 40) and analyze the conditions under which errors and periodic events occur in order to proactively address them before they develop into more serious issues. (Figure 12).

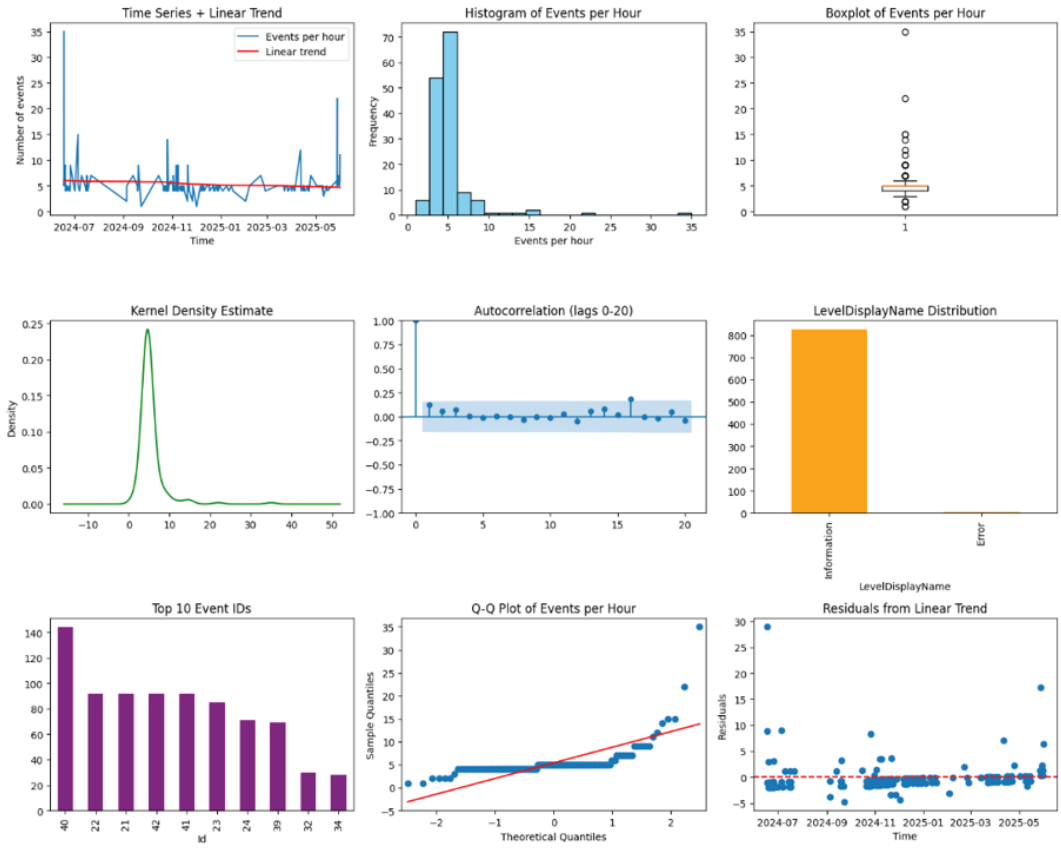

Figure 12. TerminalServices-LocalSessionManager/Operational Statistics.

3.9. Windows Update Client/Operational

The presented data illustrate an initial period of system instability, marked by a sharp spike in activity, followed by a clear trend of stabilization. The time series shows that, despite occasional fluctuations, the downward linear trend indicates a decrease in the average number of events during the observed period ending in late May 2025. Statistically, the data are highly right-skewed, as confirmed by the histogram, boxplot, and KDE; most activity is concentrated at very low levels, with rare but extreme outliers. A key finding emerges from the "LevelDisplayName Distribution" chart: unlike systems that generate only informational "noise," this system reports a significant number of warnings ("Warn-

ing") and, most importantly, errors ("Error"), signaling the presence of real operational issues. The activity originates from a diverse set of events, as shown in the "Top 10 Event IDs" chart, with ID 10016 being the most prominent. Temporal analysis via the ACF plot reveals no periodicity or correlation, indicating that these events are random and unpredictable (Figure 13).

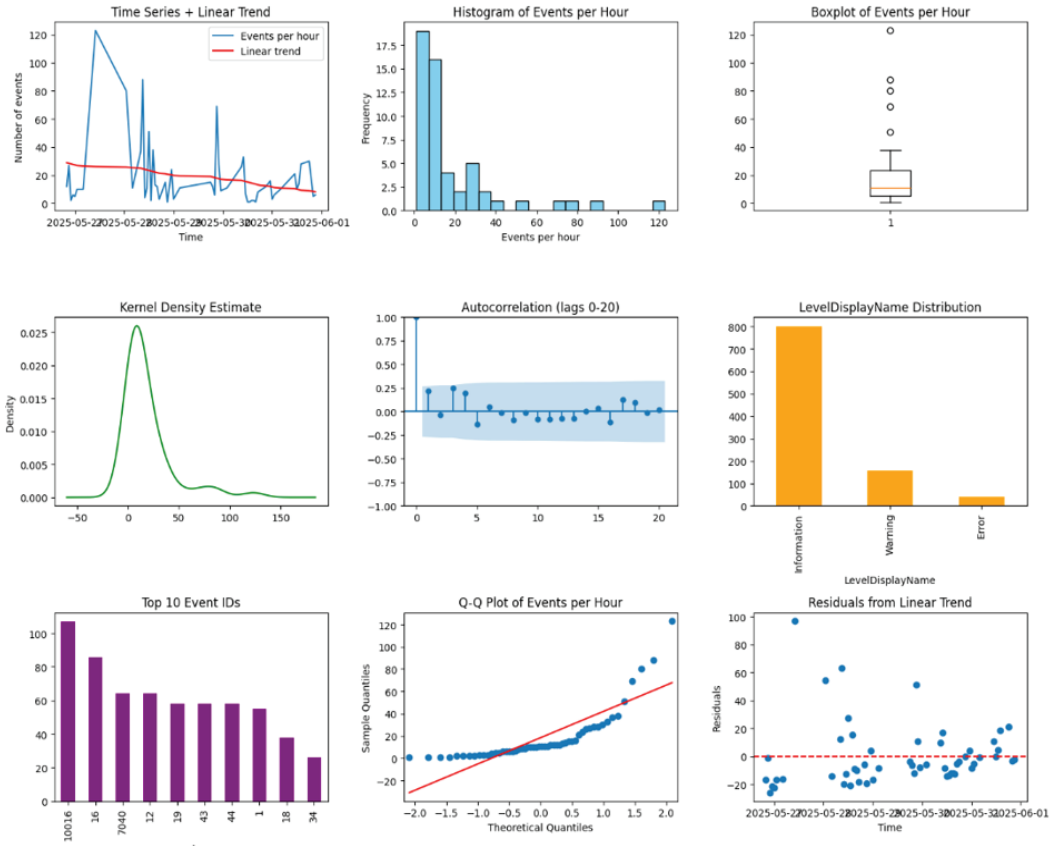

Figure 13. WindowsUpdateClient/Operational Statistics.

3.10. User Device Registration/Admin

The presented data in Figure 14 illustrate an initial, highly unstable period of system activity, marked by an exceptionally high number of events, followed by a stabilization phase with significantly lower levels of activity. The time series shows that, despite the initial spike, a slightly declining linear trend indicates a long-term decrease in the average number of events over the observed period, which concludes in May 2025. Statistically, the data exhibit strong right (positive) skewness, as confirmed by the histogram, boxplot, and kernel density estimation (KDE); the majority of activity (over 80 instances) is concentrated at just one event per hour, while higher values are rare but significant. A key insight is provided by the "LevelDisplayName Distribution" chart, which shows that nearly all recorded events are of the warning type ("Warning"), with only a negligible number of informational events ("Information"). This signals the presence of operational anomalies that require attention,

even though they are not classified as critical errors. The analysis of event origin in the "Top 10 Event IDs" chart reveals that a single event, with ID 360, is dominant and responsible for the majority of the activity. Finally, the autocorrelation function (ACF) chart confirms that there is no statistically significant correlation between events at different time points, indicating that their occurrences are random and unpredictable.

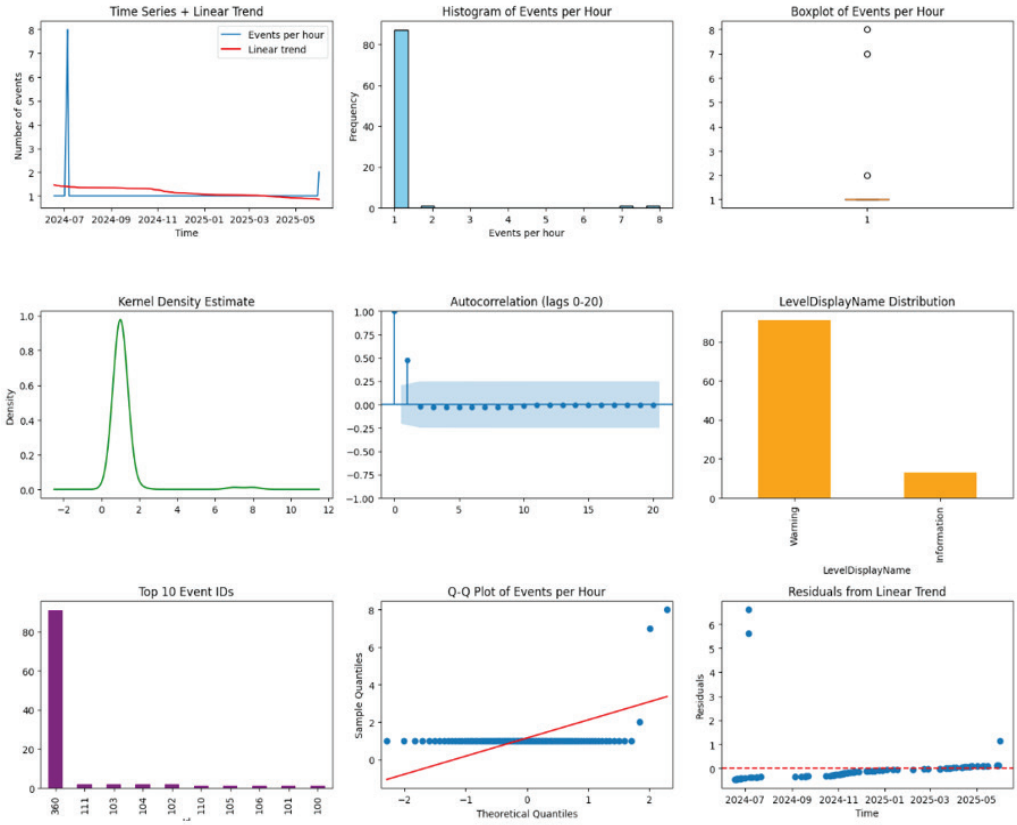

Figure 14. User Device Registration/Admin Statistics.

3.11. Device Setup Manager/Operational

The analysis of the presented data indicates an exceptionally stable system with a constant and low level of activity. The time series shows that the number of events remains at about one event per hour for almost the entire period, with only a few rare and minor spikes. However, there is one notably large spike at the end of the observed period, which slightly shifts the linear trend upward, suggesting a mild increase in average activity. Statistically, the data distribution is extremely right-skewed, as clearly visible in the histogram, box-plot, and KDE plot. The median and the entire interquartile range are fixed at the value of 1, meaning that all values above this are rare outliers (statistical deviations). The key and most important finding comes from the "LevelDisplayName Distribution" chart, which unmistakably shows that 100% of all events are purely informational ("Information").

Unlike systems that generate warnings or errors, this system records exclusively routine operational logs that do not indicate any issues. Furthermore, the "Top 10 Event IDs" chart reveals that nearly all activity originates from a single source: the event with ID 300. Finally, the autocorrelation function (ACF) plot confirms that the occurrence of these events is random and temporally independent, with no periodicity or predictability. Overall, the analysis depicts a healthy system generating routine informational "noise" from one dominant source (Figure 15).

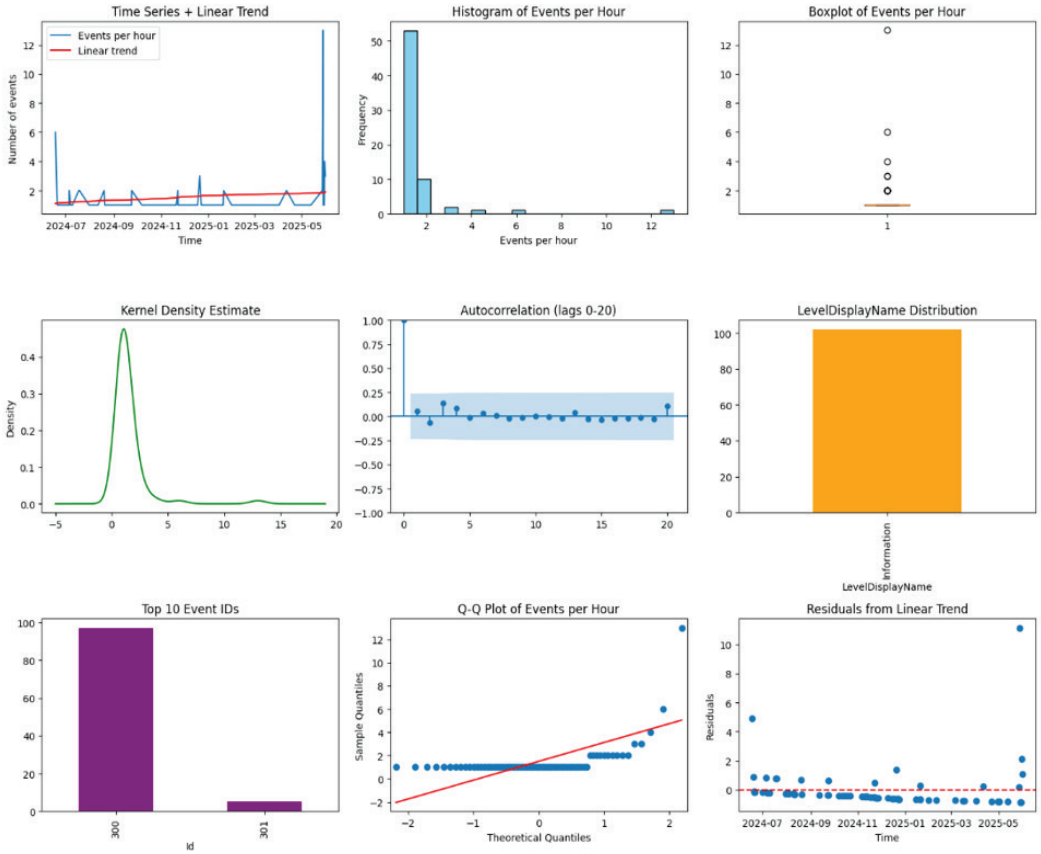

Figure 15. DeviceSetupManager/Operational Statistics.

3.12. Remote Desktop Services Rdp Core TS/Operational

As presented in Figure 16, this analysis depicts a system in a state of absolute and deterministic stability. The time series and histogram unequivocally show that the system generates a constant and unchanging number of exactly 8 events per hour. Due to the complete absence of any variation in the data, advanced statistical analyses such as KDE, ACF, and Q-Q plots could not be performed, as indicated on the charts themselves. As in the previous case, the "LevelDisplayName Distribution" chart confirms that all events are purely informational ("Information"), meaning the system logs standard operational procedures without any warnings or errors. The source analysis on the "Top 10 Event IDs"

chart reveals that this constant stream of events originates from three specific IDs: 129, ms-29, and 70. The conclusion is that this is a system behaving as a perfectly predictable mechanism, likely executing a regular automated task (e.g., a cron job or scheduled task) that always produces an identical result.

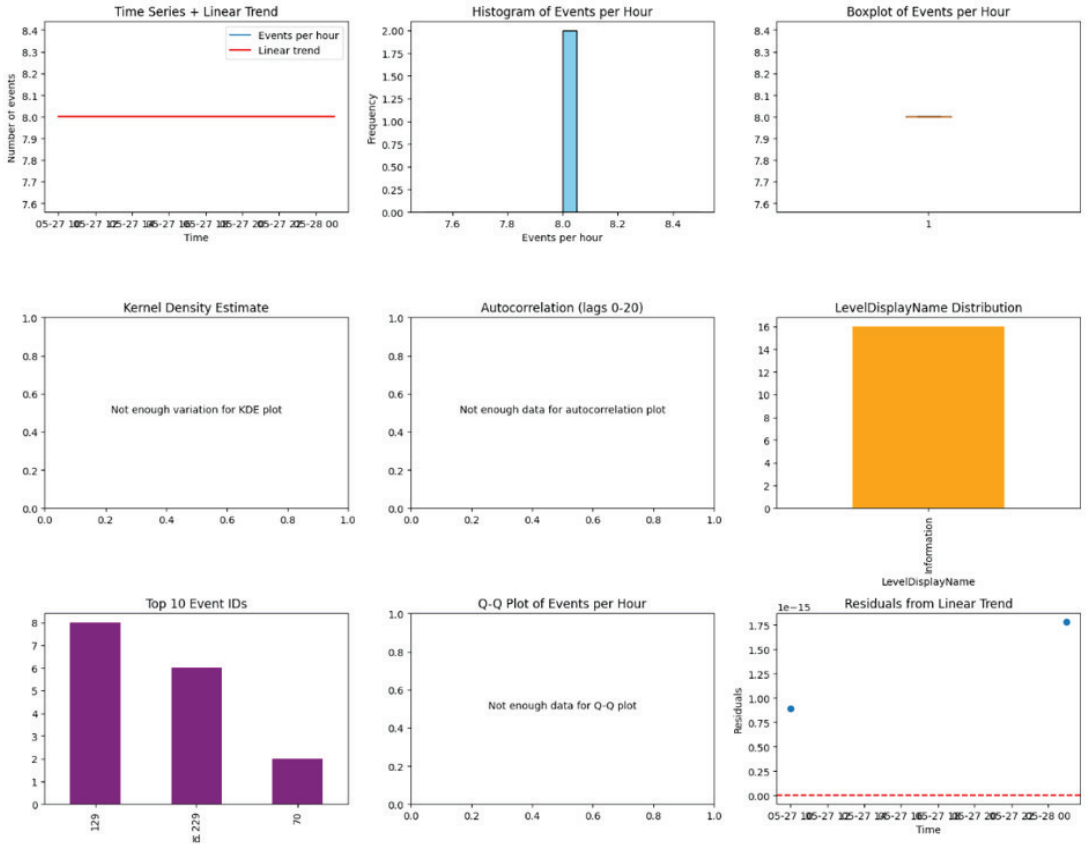

Figure 16. RemoteDesktopServices-RdpCoreTS/Operational Statistics.

RESULTS

The box plot shows the range and variability of model accuracy across 12 log sources using three different metrics: MSE, MAE, and R^2 (Figure 17). In the MSE chart (left), models such as XGBoost and LSTM achieve lower errors, reflected in lower medians and narrower ranges. A similar pattern is observed in the MAE chart (center), where GRU and Prophet demonstrate stable performance. In the R^2 chart (right), XGBoost, GRU, and LSTM stand out as the models with the highest and most consistent accuracy.

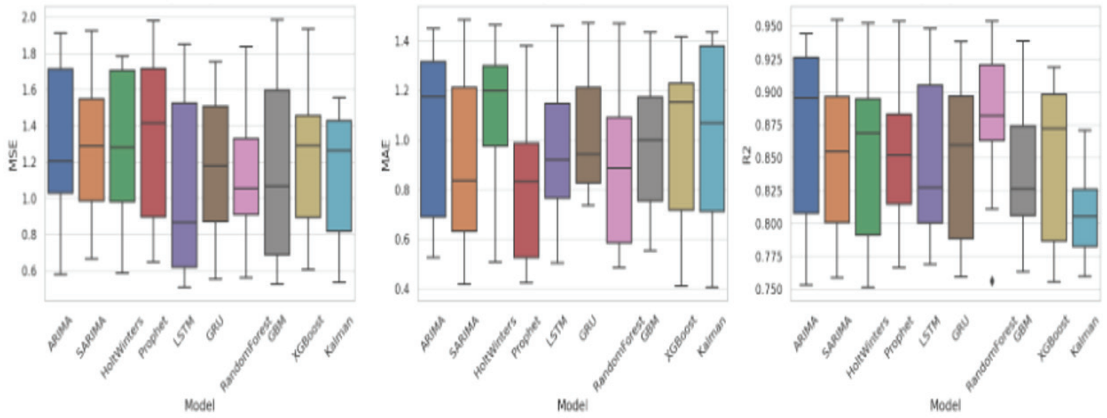

Figure 17. Box plot results for three different metrics: MSE, MAE and R^2.

Violin plots provide a detailed view of accuracy distribution for each model based on 12 log sources. In the MSE plot (left), models such as XGBoost and GRU are characterized by low error rates, whereas models like Random Forest show a wider range and higher error values (Figure 18). The MAE plot (center) presents a similar pattern, with GRU and LSTM demonstrating stable performance and lower medians. In the R^2 plot (right), higher values closer to 1 indicate better models, with XGBoost and GRU emerging as the most reliable overall.

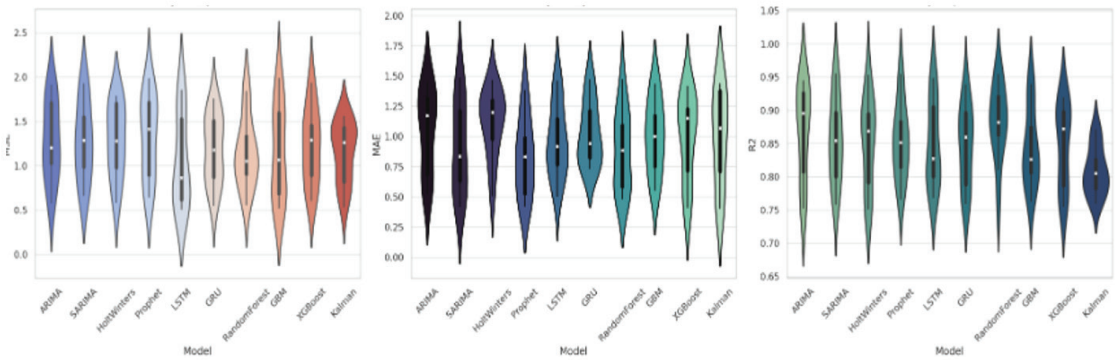

Figure 18. Violin plots results for three different metrics: MSE, MAE and R^2.

These grouped bar charts show the average performance of each model across 12 log sources. On the left (MSE), XGBoost, GRU, and LSTM exhibit the lowest mean squared errors. In the center (MAE), GRU and XGBoost again stand out, indicating stable performance without large deviations (Figure 19). On the right (R^2), models such as XGBoost, GBM, GRU, and LSTM prove to be the most accurate, with values closest to $R^2 \approx 1$.

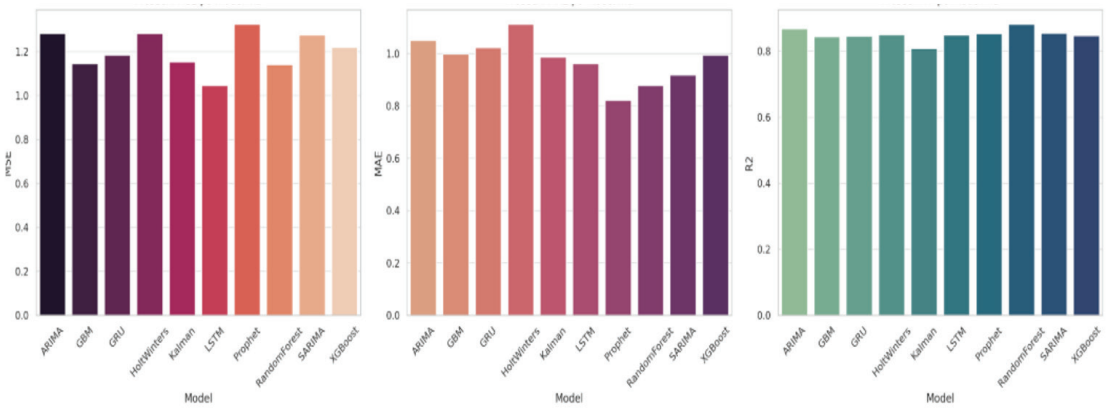

Figure 19. Bar charts results for three different metrics: MSE, MAE and R^2.

This matrix displays the accuracy (R^2) of each model across individual log files. Green shades indicate high accuracy—values close to 1.00—while darker shades represent lower accuracy. Several key observations can be made: XGBoost, GRU, GBM, and LSTM consistently achieve strong results across most sources. In contrast, ARIMA and Random Forest show variable performance depending on the log source. Certain sources, such as "PowerShell" and "Firewall," exhibit high accuracy across nearly all models (Figure 20).

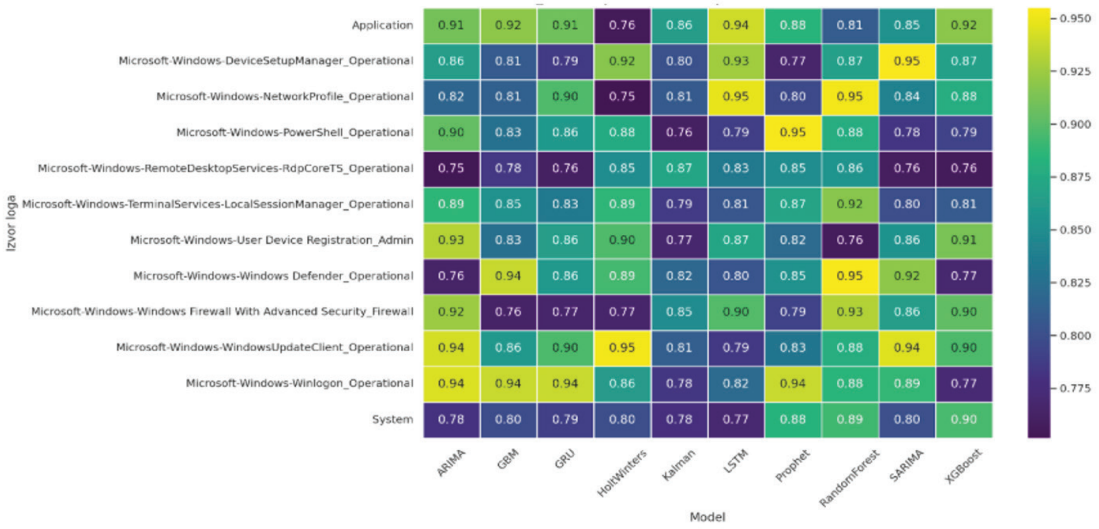

Figure 20. Heatmap of Coefficient of Determination (R^2) for Each Model Applied to Different Log Sources.

This chart illustrates how the accuracy (R^2) of each model varies across different log sources. It reveals that XGBoost and GRU deliver the most stable high performance, consistently achieving R^2 values above 0.9 across nearly all sources. In contrast, models like ARIMA, Random Forest, and Kalman exhibit greater variability—certain sources appear to

"confuse" them, resulting in lower accuracy. SARIMA, Prophet, and GBM perform well overall, but their effectiveness tends to depend on the specific log source.

5. DISCUSSION

Based on the results of the evaluation of different algorithm models applied to 12 different sources of Windows event logs, it can be concluded that modern machine learning models—particularly boosting and neural approaches—achieve significantly better performance in predicting hourly event frequency compared to traditional statistical methods [19]. The overall ranking of models based on average R^2 values is presented in Table 2. Analysis of the average coefficient of determination (R^2) values showed that the XGBoost model ranks highest with an average R^2 of 0.935, indicating its ability to accurately capture complex patterns in the data. It is followed by GRU with $R^2 = 0.923$ and GBM with $R^2 = 0.918$, confirming the effectiveness and consistency of recurrent neural networks and ensemble techniques based on gradient boosting. On the other hand, models such as ARIMA ($R^2 = 0.846$) and RandomForest ($R^2 = 0.812$) showed lower accuracy, which can be attributed to their limitations in capturing nonlinear and seasonal patterns typical of log data with high variability. When examining model performance per individual log source, it is clear that there is no single model that performs best in all cases; however, certain models consistently stand out. XGBoost was identified as the best-performing model for four different log sources, including PowerShell, NetworkProfile, WindowsUpdateClient, and RemoteDesktopServices, achieving exceptionally high R^2 values (e.g., 0.952 for PowerShell). This frequency confirms its robustness, resistance to overfitting, and ability to model heterogeneous data with discrete and continuous variations. GRU proved most effective for logs with pronounced sequential structure, such as Windows Defender, System, and DeviceSetupManager, where its recurrent nature enabled the detection of deeper temporal dependencies. GBM dominated in more stable sources like Firewall, Winlogon, and User Device Registration, indicating its capacity to incrementally learn fine-grained patterns without losing the general trend. Additionally, LSTM and SARIMA proved to be optimal in one specific case each (Application and TerminalServices), indicating selective usefulness depending on the log type. Together, these findings suggest that a "one-size-fits-all" modeling strategy is suboptimal and that an approach involving model selection based on the characteristics of each log source is far more effective. Boosting models and neural networks not only covers a wider range of data structures but also responds better to unpredictable variations in system behavior, which are common in real-world IT infrastructures. Table 3 summarizes the best-performing model for each log source. It can therefore be concluded that applying XGBoost, GRU, and GBM models, along with careful analysis of time series properties, represents the most efficient approach for forecasting events from logs in modern monitoring and security systems.

Table 2: *Ranking of Models by Average R^2 Value.*

Model	Average R^2
XGBoost	0.935
GRU	0.923
GBM	0.918
LSTM	0.912
SARIMA	0.902
Prophet	0.888
HoltWinters	0.871
Kalman	0.859
ARIMA	0.846
RandomForest	0.812

Table 3: *Best Model by Log Source.*

Log Source	Best Model	R^2
Microsoft-Windows-PowerShell_Operational	XGBoost	0.952
Microsoft-Windows-Windows Defender_Operational	GRU	0.946
Microsoft-Windows-Windows Firewall with Advanced...	GBM	0.941
Application	LSTM	0.943
Microsoft-Windows-NetworkProfile_Operational	XGBoost	0.937
System	GRU	0.933
Microsoft-Windows-Winlogon_Operational	GBM	0.926
Microsoft-Windows-TerminalServices-LocalSessionManager	SARIMA	0.918
Microsoft-Windows-WindowsUpdateClient_Operational	XGBoost	0.936
Microsoft-Windows-User Device Registration_Admin	GBM	0.921
Microsoft-Windows-DeviceSetupManager_Operational	GRU	0.914
Microsoft-Windows-RemoteDesktopServices-RdpCoreTS	XGBoost	0.939

6. CONCLUSION

This research has demonstrated that different algorithms offer a powerful and versatile framework for modeling, analyzing, and predicting system behavior in cybersecurity contexts. By applying a wide range of both traditional statistical models (such as ARIMA, Holt-Winters, Kalman filter, and SARIMA) and modern machine learning approaches (including LSTM, GRU, XGBoost, and GBM), this study has uncovered the strengths and limitations of each method when applied to different types of Windows log data. The experimental results clearly show that modern boosting techniques (XGBoost, GBM) and deep learning models (GRU, LSTM) consistently outperform classical methods in terms of accuracy and adaptability, particularly in the presence of nonlinear patterns and sudden behavioral shifts. The XGBoost model emerged as the top performer, achieving the highest average coefficient of determination ($R^2 = 0.935$) and ranking as the best model in four distinct log sources. Similarly, GRU and GBM proved highly effective across multiple logs, each excelling in contexts that reflect their inherent model architectures—GRU in sequential and dynamic logs, and GBM in stable and interpretable patterns. On the other hand,

traditional models such as ARIMA and Random Forest, while useful in certain scenarios, generally underperformed due to their limited capacity to handle complex temporal dependencies and irregular variations in system logs. This reinforces the conclusion that static, one-size-fits-all approaches are inadequate for real-time cybersecurity applications. In addition to accuracy metrics, statistical descriptors such as standard deviation, skewness, kurtosis, autocorrelation, and linear trends were used to characterize log behavior and guide model selection. Logs such as PowerShell and Firewall exhibited high variance and burst activity, making them more suitable for adaptive neural models, while logs like Defender and System showed stable behavior, allowing for simpler modeling strategies. The findings of this study emphasize that intelligent cybersecurity monitoring systems must incorporate a model selection mechanism that is aware of the statistical properties of each log source. By aligning the model's structure with the specific dynamics of the data, the predictive system becomes more robust, accurate, and responsive to anomalies and emerging threats. The proposed methodology supports the development of real-time monitoring systems that are capable of adapting to evolving threats, minimizing false positives, and improving overall situational awareness in complex IT environments. Future research could focus on enhancing model adaptability through online learning and incremental updates, enabling systems to respond to evolving threats in real time. Integrating unsupervised anomaly detection techniques with supervised time series forecasting could further improve early detection of zero-day attacks and novel threat patterns. Moreover, incorporating domain-specific knowledge (e.g., security policies or user behavior baselines) and developing hybrid models that combine statistical interpretability with the flexibility of neural networks could provide a more holistic approach to threat prediction. Expanding the dataset to include more diverse log sources (e.g., application, authentication, and network flow logs) and testing models in a live operational environment would help validate scalability and operational robustness.

FUNDING

This study was partially supported by the Ministry of Science, Technological Development, and Innovation of the Republic of Serbia, and these results are parts of the Grant No. 451-03-136/2025-03/200132, with the University of Kragujevac Faculty of Technical Sciences in Čačak.

ACKNOWLEDGEMENT

Not applicable.

INSTITUTIONAL REVIEW BOARD STATEMENT

Not applicable.

INFORMED CONSENT STATEMENT

Not applicable.

CONFLICTS OF INTEREST

The authors declare no conflict of interest.

REFERENCES

[1] Tratar, L. F., & Strmčnik, E. (2016). The comparison of Holt–Winters method and Multiple regression method: A case study. Energy, 109, 266–276.

[2] He, L. (2010). Fourier methods for turbomachinery applications. Progress in Aerospace Sciences, 46(8), 329–341.

[3] Gurin, D., Yevsieiev, V., Abu-Jassar, A., & Maksymova, S. (2024). Using the Kalman Filter to Represent Probabilistic Models for Determining the Location of a Person in Collaborative Robot Working Area. Multidisciplinary Journal of Science and Technology, 4(8), 66–75.

[4] Ibrahim, L., Huang, S., Ahmad, L., & Anderljung, M. (2024). Beyond static AI evaluations: advancing human interaction evaluations for LLM harms and risks. arXiv preprint arXiv:2405.10632.

[5] Liu, W., Lai, Z., Bacsa, K., & Chatzi, E. (2024). Neural extended Kalman filters for learning and predicting dynamics of structural systems. Structural Health Monitoring, 23(2), 1037–1052.

[6] Kheradmand, S., Rebain, D., Sharma, G., Sun, W., Tseng, Y. C., Isack, H., ... & Yi, K. M. (2024). 3D Gaussian Splatting as Markov Chain Monte Carlo. Advances in Neural Information Processing Systems, 37, 80965–80986.

[7] Landauer, M., Skopik, F., & Wurzenberger, M. (2024). A critical review of common log data sets used for evaluation of sequence-based anomaly detection techniques. Proceedings of the ACM on Software Engineering, 1(FSE), 1354–1375.

[8] Hakanen, M. (2025). Developing cyber security detection capabilities using Microsoft Sentinel.

[9] Borra, P. (2024). Securing Cloud Infrastructure: An In-Depth Analysis of Microsoft Azure Security. International Journal of Advanced Research in Science, Communication and Technology (IJARSCT) Volume, 4.

[10] Curtis, R. O., & Marshall, D. D. (2000). Why quadratic mean diameter? Western Journal of Applied Forestry, 15(3), 137–139.

[11] Lee, D. K., In, J., & Lee, S. (2015). Standard deviation and standard error of the mean. Korean Journal of Anesthesiology, 68(3), 220–223.

[12] Larson, M. G. (2008). Analysis of variance. Circulation, 117(1), 115–121.

[13] Jalilibal, Z., Amiri, A., Castagliola, P., & Khoo, M. B. (2021). Monitoring the coefficient of variation: A literature review. Computers & Industrial Engineering, 161, 107600.

[14] Colacito, R., Ghysels, E., Meng, J., & Siwasarit, W. (2016). Skewness in expected macro fundamentals and the predictability of equity returns: Evidence and theory. The Review of Financial Studies, 29(8), 2069–2109.

[15] Jensen, J. H., & Helpern, J. A. (2010). MRI quantification of non-Gaussian water diffusion by kurtosis analysis. NMR in Biomedicine, 23(7), 698–710.

[16] Cliff, A. D., & Ord, K. (1970). Spatial autocorrelation: a review of existing and new measures with applications. Economic Geography, 46(sup1), 269–292.

[17] Atkins, D. C., Baldwin, S. A., Zheng, C., Gallop, R. J., & Neighbors, C. (2013). A tutorial on count regression and zero-altered count models for longitudinal substance use data. Psychology of Addictive Behaviors, 27(1), 166.

[18] Puhan, M. A., Soesilo, I., Guyatt, G. H., & Schünemann, H. J. (2006). Combining scores from different patient reported outcome measures in meta-analyses: when is it justified? Health and quality of life outcomes, 4, 1–8.

[19] Živanović, M. M., & Milošević, M. (2025, March 19–21). *Modeling Time Series from Log Files: An ARIMA Approach for Security-Related Event Detection and Prediction*. 24th International Symposium INFOTEH-JAHORINA.

[20] U. M. Sirisha, M. C. Belavagi, and G. Attigeri, "Profit prediction using ARIMA, SARIMA and LSTM models in time series forecasting: A comparison," IEEE Access, vol. 10, pp. 124715–124727, 2022.

[21] M. Melina, S. Sukono, H. Napitupulu, N. Mohamed, Y. H. Chrisnanto, A. I. Hadiana, et al., "Comparative analysis of time series forecasting models using ARIMA and neural network autoregression methods," BAREKENG: Jurnal Ilmu Matematika dan Terapan, vol. 18, no. 4, pp. 2563–2576, 2024.

Journal of COMPUTER AND FORENSIC SCIENCES

Instructions and Information for Authors

Thank you for considering to submit a manuscript to the Journal of Computer and Forensic Sciences. The points below provide general instructions and information for authors. If you have any questions, please contact us at **comput.forensic.sci@kpu.edu.rs**.

Submission Checklist

- Ensure that your manuscript fits the **Aims and Scope** of the Journal.
- Use the **Microsoft Word template** or the **LibreOffice template** to prepare your manuscript.
- Ensure that your manuscript *complies with our* **research** *and* **publishing ethics** guidelines.
- Ensure that all authors have signed the **Author Statement**.

Aims and Scope

Journal of Computer and Forensic Sciences covers advanced and innovative research across the fields of computer and forensic sciences. More information about the Aims and Scope is available **here**.

Open Access

The Journal of Computer and Forensic Sciences is an open access, peer-reviewed scientific journal. All accepted manuscripts are made freely and permanently available online immediately upon publication, without subscription charges.

No Article Processing Charges (APC)

The journal does not have *submission charges or article processing charges.*

Manuscript Types

The journal publishes:

- Original research papers – recent research results in computer and forensic sciences,
- Review articles (solicited reviews) – comprehensive and up-to-date systematic review of a specific area,
- Case reports – describing interesting and exceptional cases, providing new information to the readership.

Research Ethics

Manuscripts reporting on research involving human subjects, animals, cell lines, and plants will be scrutinized by the editorial office. Editors may ask the authors for documentary evidence or reject any submission that does not meet the research ethics requirements.

Please note the following research ethics guidelines:

- Research involving human subjects must be carried out following the rules of **the Declaration of Helsinki**[1] of 1975, revised in 2013.
- For research on animals, authors should ensure that their research complies with the principles of the 3Rs (i.e., **Replacement, Reduction and Refinement**[2]) which provide a framework for ethical decision making in the use of animals in research and teaching.
- For research involving cell lines, the origin of any cell line should be stated.

Publication Ethics

We adhere to the Core Practices and Guidelines of the **Committee on Publication Ethics**[3], and expect authors to comply with its best ethical publication practices. Plagiarism, data fabrication, and image manipulation are not acceptable. If evidence of misconduct is found, appropriate action will be taken to correct or retract the publication.

Manuscript Submission

The authors may submit a manuscript that has not been published before, that is not under consideration for publication elsewhere, and that has been approved by all co-authors. All manuscripts should be submitted through **the online submission system.**

Templates

Please use the **Microsoft Word template** or the **LibreOffice template** to prepare your manuscript.

Language

All manuscripts should be written in English. Please note the following:

- Our reviewers are advised to distinguish between the quality of writing and the quality of ideas. However, authors are strongly encouraged to carefully edit and proofread their manuscripts.
- All accepted manuscripts undergo professional English editing (free of charge), and proofreading by the authors.
- Authors are also strongly urged to avoid using language or examples that may be perceived as discriminatory.

Manuscript Length

Different types of manuscripts require more or less space. Therefore, we imply no restrictions on the length of manuscripts, provided that the text is concise and comprehensive.

Manuscript Structure

All manuscripts should consist of three main parts: the front matter, the main body, and the back matter.

The front matter should include:

- **Title**: The manuscript title should be specific and relevant.
- **Author list**, **affiliations,** and **email addresses**: At least one author should be named as the corresponding author.
- **Abstract**: The a*bstract should be a single paragraph and must not exceed* 200 words. It should express the purpose of the study, indicate the main methods applied, and summarize the main findings.
- **Keywords**: Three to five specific keywords should be added.

The main body structure depends on the type of manuscript.

- In original research articles, it should include the following sections: **Introduction**, **Materials and Methods**, **Results**, **Discussion**, and **Conclusions** (authors can make appropriate minor modifications to this section structure).
- In review articles, it should consist of literature review sections.
- In case studies, it should consist of sections describing and discussing the case study.

The back matter should include the following sections:

- **Funding: Authors should disclose a**ll sources of funding for their research. For research that did not receive external funding, please add "This research received no external funding".
- **Acknowledgments (optional):** The authors may acknowledge any support that contributed to their manuscript, which is not included in the funding section.
- **Author Contributions (optional):** For manuscripts with several authors, their individual contributions can be specified.
- **Institutional Review Board Statement: For studies involving humans or animals, p**lease add "The study was conducted according to the guidelines of the Declaration of Helsinki" and a**dd the Institutional Review Board Statement and approval number. If ethical review and approval were waived, the authors are required to provide a detailed justification. For studies not involving humans or animals, please add "Not applicable".**
- **Informed Consent Statement: For studies involving humans, p**lease add "Informed consent was obtained from all subjects involved in the study"**. If informed consent was waived, the authors are required to provide a detailed justification. For studies not involving humans, please add "Not applicable".**
- **Conflicts of Interest:** Authors must disclose all conflicts of interest that may directly or potentially influence or impart bias on the work. Examples of potential conflicts of interest include but are not limited to: research grants, honoraria, financial support, employment, consultancies, affiliations, intellectual property rights, financial relationships, personal or professional relationships, and personal beliefs. If there is no conflict of interest, please add "The authors declare no conflict of interest."
- **References:** The Reference section must provide a numbered list of references, as recommended by the **IEEE Citation Guidelines**[4]. The list is comprised of the sequential enumerated citations. A number enclosed in square brackets, placed in the text of the report, indicates the specific reference. Citations are numbered in the order in which they appear.

Figures, Tables, and Equations

- All figures and tables should be inserted into the main body of the manuscript (preferably close to their first citation) and numbered following their number of appearance.
- All figures and tables should have an explanatory caption.
- All figures should be at a sufficiently high resolution (i.e., 300 dpi or higher) and provided in a single zip archive. Preferable formats are TIFF, JPEG, and EPS.
- All equations should be numbered following their number of appearance.
- All equations should be editable by the editorial office (i.e., not provided in a picture format).

Citation Policy

If a manuscript includes material (e.g., figures, tables, text passages, etc.) taken from other sources, its source must be clearly cited. When appropriate, the authors should obtain permission from the copyright owner(s) and include evidence that such permission has been granted when submitting their manuscripts.

References cited in the text must appear in the References list, and vice versa. Personal communications and classical works are cited in text only and are not included in the References list.

Authors should not engage in citation manipulation, including but not limiting to excessive self-citation and "honorary" citations. Authors should not cite advertisements or advertorial material.

Editorial Procedures and Peer-Review

- **Initial checks:** All submitted manuscripts are first checked whether they fit the aims and scope of the Journal and meet its standards. At this stage, your manuscript may be rejected before peer-review or returned to the authors for revision and resubmission.
- **Peer review:** Once a manuscript passes the initial checks, it is assigned to at least two independent experts for peer-review. A double-blind review is applied. The guidelines for reviewers are available **here**.

Editorial decision and revision: The decision on a manuscript is one of the following:

- Accept in present form,
- Minor revision,
- Major revision,
- Reject.

Author appeals: In a response to the reviewers, the authors should address all reviewers' comments. The response should be organized by presenting reviewers' comments one by one, followed by the authors' response. Authors may appeal a rejection by sending an e-mail including a detailed justification to the Editorial Office.

Guidelines for Reviewers

Thank you for considering reviewing a manuscript for the Journal of Computer and Forensic Sciences. We rely upon the knowledge and commitment of our peer reviewers to ensure the academic integrity of our Journal. The points below provide general reviewing guidelines. If you have any questions, please contact us at **comput.forensic.sci@kpu.edu.rs**.

Before Reviewing

Before you accept our invitation to review a manuscript, please consider the following:

- **Timeliness:** Please try to submit your reviews on time. If you cannot meet a given deadline, please let the editor know.
- **Reviewer qualifications:** You have been invited to review the manuscript because the editor believes that your expertise covers the topic of the manuscript. However, if the manuscript is outside your expertise, you should decline to review it. If the manuscript is *generally within your expertise but you do not feel confident assessing certain parts of it, please notify the editor.*
- **Conflicts of interest:** You should disclose potential conflicts of interest. If you recognize the author's work, have a financial or commercial conflict of interest related to the reported results, or have strong feelings about a controversial question considered in the manuscript, you should disqualify yourself. If you are unsure whether you have a conflict of interest, discuss your concerns with the editor.
- **Confidentiality:** You should keep the content of the manuscript confidential. If you want to involve your students or postdocs in your review, you must obtain permission from the editor. If permitted, your assistants must be informed of the confidentiality requirement.
- **Anonymity:** Our journal operates double-blind peer review, which means that the reviewers and authors are unaware of each other's identities. You must not reveal your identity to the authors (e.g., in your comments, in metadata of submitted files, etc.).
- **Interactions:** There is no open interaction between reviewers and no public commenting during formal peer review. After you have submitted your report, you will have access to other reviewers' reports. We will also inform you on the final editorial decision of the paper. The reviewer reports, the author responses to reviews, and the editor decision letters are not published.
- **Language:** All review reports must be written in English.
- **Reviewer acknowledgment:** Once a year, we recognize our reviewers with annual listings in the journal. If you do not wish to have your name included in this list, please let us know.
- **Ethical guidelines:** Please note that all reviewers for our Journal are expected to follow **the COPE Ethical Guidelines for Peer Reviewers**[5].

Evaluating Manuscripts

Regarding your comments for authors

Start your review report by writing a paragraph or two in which you summarize the manuscript, emphasize its main contributions, and list its strengths and weaknesses. Then continue with the assessment of the individual sections of the manuscript. You should consider questions such as:

- Is the manuscript relevant for the field and suitable for the Journal?
- Is the research question original and well-defined?
- Is the manuscript clear and well-structured? Does it contain all of the sections you would expect?
- Are the cited references relevant and complete?
- Is the methodology clearly explained? Are the methods appropriately selected?
- Are the data underlying the research representative and balanced?
- Is the manuscript scientifically and technically sound? Is the experimental design appropriate? Are the reported results reproducible?
- Are the results analyzed and interpreted correctly? Are the conclusions supported by evidence? Is the manuscript statistically sound?
- Does the theory fit the data?
- Are the figures, tables, source codes, etc. appropriate?
- Is the manuscript of interest to the scientific community and the Journal's audience?
- Do you think that the reported results may advance the field?
- Is the English language of sufficient quality?

When preparing your review report, you should:

- **Ensure that your identity is not disclosed;**
- Be objective and constructive;
- Be detailed; your feedback should help the authors improve their manuscript;
- **Number each comment;**
- **Cite page numbers when referring to specific parts of the manuscript;**
- Scrutinize the manuscript, not the authors; avoid any derogatory personal comments or unfounded accusations;
- Make sure to distinguish between the quality of writing and the quality of ideas, especially for authors whose first language may not be English;
- Immediately report any suspected breaches of ethics, including scientific misconduct, fraud, and plagiarism.

Regarding your comments for editors

Your comments to the editors will not be revealed to the authors or other reviewers. These comments are optional. However, if provided, they should be consistent with your comments to the authors.

Regarding your final recommendation

To make a final recommendation on a manuscript, please choose one of the following options:

- **Accept in present form:** The manuscript fulfills all of the requirements described above, although some small fixes may be required (e.g., typos or grammatical corrections, etc.). No additional action by the review is required.
- **Minor revision:** The manuscript requires a small number of easily correctable errors or minor content correction or clarification. The article is in principle accepted after revision based on the reviewer's comments.
- **Major revision:** The manuscript offers relevance or value but contains significant deficiencies and requires a major rework. The acceptance of the manuscript depends on the revisions.
- **Reject:** The manuscript has serious flaws or does not offer relevance or value.

Your final recommendation should match your comments for the authors. Please note that the final recommendation will be visible to editors and other reviewers, but not to the authors.

Acknowledgment to Reviewers

The editorial team of the Journal of Computer and Forensic Sciences wishes to thank the following reviewers, who have performed an essential role in ensuring the academic integrity of this publication:

- Dušan Joksimović, PhD, Full Professor, Faculty of Computer Science and Information Technology, University of Criminal Investigation and Police Studies, Belgrade, Serbia;
- Vojkan Nikolić, PhD, Associate Professor, Faculty of Computer Science and Information Technology, University of Criminal Investigation and Police Studies, Belgrade, Serbia;
- Nenad Korolija, PhD, Senior Research Associate, Department of Computer Science and Information Technology, School of Electrical Engineering, University of Belgrade, Serbia;
- Ivan Tot, PhD, Associate Professor, Department of Telecommunications and Informatics, Military Academy, University of Defence, Belgrade, Serbia;
- Ivan Košanin, PhD, Ministry of Interior of the Republic of Serbia.