A Next-Generation Deep Learning-Powered Security Operations Center (SOC) for Online Learning Platforms

Ali Radid[1], Mohamed Ghazouani[1*] and Benlahmar Habib[1]

[1] Hassan II University of Casablanca. 19, Tariq Bnou Ziad, Morocco
*ghazouani_mohamed@yahoo.fr

Abstract. The digital landscape is experiencing a concerning surge in online threats, with their associated risks growing exponentially, especially within the realm of online learning platforms. These threats span a broad spectrum, ranging from sophisticated cyberattacks to pervasive phishing campaigns and insidious malware. In an era dominated by digitalization, no entity operating within online learning environments is immune to the reach of these threats. Recognizing the urgency of the situation is crucial, as inaction in the face of this escalating danger could have severe consequences, impacting not only the economy but also the privacy and security of individuals engaged in online learning. Effectively addressing the mounting threat landscape requires a multifaceted approach. Solutions must encompass advanced cybersecurity measures, robust threat intelligence, proactive risk management, and comprehensive awareness and training initiatives tailored to online education platforms. This article introduces an innovative approach to enhance Security Operations Centers (SOCs) specifically within the context of online learning platforms, harnessing the power of deep learning. By integrating cutting-edge deep learning techniques into the SOC framework, the goal is to significantly enhance the capability to detect, analyze, and respond to complex cybersecurity threats within the unique challenges of online education. This groundbreaking model leverages artificial intelligence to provide real-time threat intelligence, predictive analysis, and anomaly detection, thereby reinforcing the security posture of online learning platforms and fortifying their defenses against a rapidly evolving threat landscape.

Keywords: Online learning platforms, Security Operations Center (SOC), Cybersecurity, Deep Learning.

1. Introduction

In an era defined by the omnipresence of digital technologies and the relentless advance of cyberspace, the need for safeguarding information and protecting critical assets has never been more pronounced. This imperative has given rise to the Security Operations Center, commonly known as the SOC. The establishment of a Security Operations Center (SOC) is designed to address various issues and enhance an organization's overall security posture. Some of the key problems a SOC aims to solve include:

- **Early Threat Detection:** SOC helps in the early detection of cybersecurity threats, such as malware, intrusions, and data breaches, ensuring that potential issues are identified before they can cause significant damage.
- **Rapid Incident Response:** A SOC enables organizations to respond swiftly and effectively to security incidents. It ensures that incidents are managed efficiently, minimizing their impact and reducing downtime.
- **Continuous Monitoring:** SOC provides 24/7 monitoring of networks, systems, and applications, ensuring that security teams have real-time visibility into potential threats and vulnerabilities.
- **Threat Intelligence:** SOC gathers and analyzes threat intelligence, allowing organizations to stay informed about emerging cyber threats and adapt their security measures accordingly.
- **Compliance Management:** Many industries have regulatory requirements for data security. A SOC helps organizations maintain compliance by monitoring security controls and providing documentation for audits.
- **Resource Optimization:** SOC optimizes the allocation of cybersecurity resources by focusing on critical areas of concern, thereby ensuring that security efforts are both effective and efficient.
- **Risk Reduction:** By proactively identifying and mitigating security threats, a SOC reduces the risk of data breaches, financial losses, and reputational damage.
- **Incident Documentation:** SOC teams maintain detailed incident records and reports, which can be crucial for forensic analysis, legal purposes, and improving security processes.
- **Security Awareness:** SOC helps improve overall security awareness within an organization by educating employees about security best practices and the importance of cybersecurity.

M. Khaldi et al. (eds.), *Proceedings of the E-Learning and Smart Engineering Systems (ELSES 2023)*, Atlantis Highlights in Social Sciences, Education and Humanities 14,
https://doi.org/10.2991/978-94-6463-360-3_29

Adaptation to Evolving Threats: As cyber threats evolve, a SOC continually updates its strategies and tools to stay ahead of attackers and protect against emerging risks.

In this paper, we are advocating for the integration of deep learning technologies within our Security Operations Center (SOC) to significantly bolster its effectiveness. Deep learning, a subset of artificial intelligence, offers the capacity to process and analyze vast amounts of data with a level of nuance and pattern recognition that surpasses conventional methods. By harnessing deep learning capabilities, our SOC will be better equipped to identify complex and evolving cyber threats, enabling faster response times and more precise threat mitigation. This strategic integration represents a pivotal step towards a more resilient and adaptive security infrastructure that can adeptly navigate the intricacies of today's cybersecurity landscape.

The rest of the paper is organized as follows. Section 2 describes related work. We review the methodology in Section 3. Theoretical explanations and experimental results of the proposed BI-LSTM model are denoted in section 4. The conclusion of the proposed model is represented in Section 5.

2. Related Work

The paper [1] suggests that a generic and effective log anomaly detection system named LogAD which is an LSTM-based approach, integrating a knowledge base and multiple anomaly detection techniques. LogAD contains four components, data preparation, log preprocessing, log anomaly detection, and alerts with visualization. They first collect unstructured raw logs to prepare for anomaly detection. The goal of log preprocessing is to transform raw logs into structured data. Afterwards, appropriate anomaly detection algorithms are selected based on the configurations in the knowledge base or engineers' selection. After identifying anomalies, LogAD generates an alert with an interpretable report, which could assist engineers in understanding this alert and gaining insights to diagnose the problem. The average F1-score of LogAD achieves 0.83, outperforming baseline approaches.

In this paper [2], more practical log-based anomaly detection approach is proposed, called PLELog, by combining the strengths of unsupervised/semi-supervised approaches (getting rid of time-consuming manual labeling) and supervised approaches (incorporating the knowledge on historical anomalies). More specifically, PLELog is a semi-supervised approach only knowing the labels of a part of normal log sequences in a training set. PLELog consists of three stages, i.e., semantic embedding, probabilistic label estimation, and anomaly detection model building. In the probabilistic label estimation a clustering algorithm was use (**HDBSCAN**) after that it comes the anomaly detection model which is based on GRU neural network. PLELog perform well on both HDFS and BGL datasets. The F1-score of PLELog is 0.957 and 0.982 on HDFS and BGL, respectively. The results demonstrate the effectiveness of PLELog.

The article [3] focuses on anomaly detection within logs in large-scale online systems, particularly in Cloud computing platforms. Anomaly detection is crucial to ensure high availability and security of client services. The authors propose MoniLog, an autonomous system for anomaly detection within logs, designed to handle Cloud computing constraints and large-scale online platforms. MoniLog consists of three main steps: log parsing, anomaly detection, and anomaly classification.
Approaches and Evaluation Metrics: The authors compare different anomaly detection approaches. They highlight that simple keyword matching and regular expression-based methods are insufficient to identify anomalies effectively, especially sequential anomalies. Instead, deep learning-based approaches like DeepLog, LogAnomaly, and LogRobust that use Long-Short-Term Memory (LSTM) neural networks have shown promise. LogRobust utilizes semantic vectorization to handle log template instability, while LogAnomaly matches new templates to existing ones for identification.
The evaluation of these approaches involves using altered versions of datasets with varying proportions of unstable log events. Metrics such as precision, recall, and F1-score are used to compare the performance of these models. However, the evaluation poses challenges, as creating datasets with a significant number of real anomalies is difficult and injecting anomalies into training data can lead to performance degradation in real-life scenarios.

The article [4] proposes an approach called BERT-Log which can detect log anomalies automatically based on BERT pre-trained and fine-tuned model. The key features of the approach are as follows:

- Log Sequence as Natural Language Sequence: BERT-Log treats the log sequence as a natural language sequence, allowing it to utilize pre-trained language models to learn the semantic representation of both normal and anomalous logs.

- Log Parsing and Feature Extraction: Raw logs are parsed into structured event templates using the Drain method. Log sequences are then obtained using a sliding window approach, which includes node ID, window size, and step size for the BGL dataset.
- Log Semantic Encoding: Log sequences are converted into embedding vectors using a pre-trained model, and semantic vectors are obtained by considering the position and segment information of tokens in the log sequence.
- Anomaly Detection: A fully connected neural network is utilized to fine-tune the BERT model for anomaly detection. Supervised learning is used to learn the semantic representation of normal and anomalous logs.

The BERT-Log approach was evaluated on two public log datasets: HDFS and BGL. The evaluation results are as follows:

- HDFS Dataset, BERT-Log achieved the highest performance among all methods with an F1-score of 99.3%. The BGL Dataset, BERT-Log detected anomalies with an F1-score of 99.4%, outperforming other related approaches such as LogRobust and HitAnomaly.
- Limited Data Usage, The model achieved high accuracy and generalization ability even with only 1% of the dataset on both HDFS and BGL datasets.
- Smaller Parameters, The proposed BERT-Log approach has smaller parameters compared to other related works, making it more efficient.

The article [5] presents an approach called LogCAD (Log-based Conformal Anomaly Detection) for log-based anomaly detection. The main components of LogCAD are data preprocessing, conformal prediction, and anomaly detection. The key steps involved in the process are as follows:
Data Preprocessing: The original log data is preprocessed to form a feature vector matrix A through log parsing and feature extraction. Logs are collected from large-scale systems, and log parsing is performed to extract event templates and specific parameters from each log message. Feature extraction is then applied to obtain valuable features from log events, and an event count matrix A is constructed to record the occurrence times of each log event in the log sequences.
Conformal Prediction: The conformal prediction framework is used to calculate the nonconformity score for each log sequence. Multiple ensemble algorithms, such as AdaBoost and Random Forest, are utilized as the nonconformity measure module. The conformal score alpha (α) is obtained, and the significance level ε is adjusted to predict the labels of the test log sequence based on the p-values obtained from the nonconformity scores.
Anomaly Detection: For anomaly detection, the LogCAD model uses a multialgorithm collaborative decision-making approach, such as voting, to reduce misjudgment and improve accuracy. The prediction set consists of multiple labels with confidence, and the final label with confidence is determined using the voting rule. Feedback mechanisms are established to handle unstable logs, where labels, nonconformity scores, and confidence are used to mitigate the impact of unstable logs and improve the model's robustness.

ConAnomaly, in paper [6] is a framework for anomaly detection in large-scale system logs, was proposed. The log content was extracted and transformed into a vector sequence using the log2vec method, inspired by word2vec. The logs were then processed using a multi-layer LSTM to learn sequential relationships and perform anomaly detection. The model was evaluated on the BGL and HDFS datasets, achieving good performance in detecting anomalies. Sensitivity to certain hyperparameters and the presence of a single category prediction phenomenon were observed, warranting further investigation. ConAnomaly was evaluated on two datasets: the BGL dataset and the HDFS dataset. Overall, ConAnomaly demonstrated promising performance in detecting anomalies in large-scale system logs, especially when utilizing the log semantic information.

This article [7] presents an unsupervised anomaly detection method for large-scale systems, such as cloud-based online service systems, where accurate and timely anomaly detection is crucial. The proposed method utilizes system logs, which contain rich information, to detect anomalies automatically. It addresses the challenges of ignoring semantic meaning in log sequences and the presence of unstable log events in real-world scenarios.
Algorithm Used: The proposed method combines semantic embedding and a CNN-LSTM model for anomaly detection. It first extracts keywords from log templates using TF-IDF and then converts them into semantic vectors through Glove word vectorization. The log count vectors are used to represent the quantitative features. The CNN captures short sequence relationships from the semantic vector sequences, and LSTM learns quantitative patterns from the log count vectors.
The proposed method was evaluated on stable and unstable log datasets. On the stable datasets (HDFS and BGL), it outperformed existing methods (PCA, LogCluster, DeepLog, and LogAnomaly) with an F-measure of 0.98 and 0.954, respectively. On the unstable log dataset, the proposed method achieved an F-measure of 0.95 even when

the injection ratio of random log modifications increased, demonstrating its robustness. Table 1 serves as a comprehensive visual representation of the literature review undertaken in the context of this article.

Table 1. Critical Review of Prior Studies.

Ref	Proposal	Drawbacks
[1]	LogAD	• Single algorithm is usually not a panacea in practice. • Choose appropriate logs for anomaly detection. • They did not study the sensitivity of parameters.
[2]	HDBSCAN/GRU	HDBSCAN algorithm used may not always produce perfect results.
[3]	LSTM	• Drain's accuracy is influenced by preprocessing and can affect the accuracy if not well defined. • Drain use two hyper-parameters and their values have a significant impact on precision. Therefore, Drain cannot be deployed in an unknown system with a high level of confidence.
[4]	BERT	• Converting raw logs into structured event templates automatically and accurately remains a challenge, especially for large volumes of logs with new formats. • Ensuring that semantic information from log sequences is effectively described and includes context and position information remains a challenge. • Properly defining sliding window parameters for logs from different nodes and over long periods is still an open challenge.
[5]	AdaBoost and Random Forest	Retraining too many times will exhaust resources and make systems too slow to learn new information in time.
[6]	Multi-layer LSTM	• Limitations in handling unseen log types in the data. • The model presented the phenomenon of predicting a single category (e.g., all logs as normal) in some cases, which was observed when the training data accounted for 10%. The authors acknowledged this limitation and intended to investigate it further.
[7]	CNN-LSTM	The model was time consuming during prediction phase.

3. Methodology

To address the limitations inherent in existing research, we have endeavored to develop a ground-breaking next-generation SOC underpinned by the innovative BI-LSTM (Bidirectional Long Short-Term Memory) architecture. This pioneering SOC represents a quantum leap in the realm of cybersecurity, seeking to address the evolving and increasingly sophisticated threat landscape. To devise an effective solution, we conducted an exhaustive evaluation of various neural network architectures, including BI-LSTM, Convolutional Neural Networks (CNN), and Gated Recurrent Units (GRU), to determine the optimal choice for our objectives. After rigorous experimentation and meticulous performance assessment, the BI-LSTM model emerged as the superior candidate, surpassing the others in terms of accuracy, adaptability, and real-time threat detection capabilities. This selection underscores our unwavering commitment to pushing the boundaries of security infrastructure, delivering a SOC that not only mitigates the limitations of past methodologies but also stands as a beacon of innovation and resilience in safeguarding critical digital assets against contemporary cyber threats. Fig. 1 illustrates the workflow of our proposed model, providing an overview of its structure and functionality. In the following section, we will provide a detailed exploration of the selected models, delving into their specific characteristics and components.

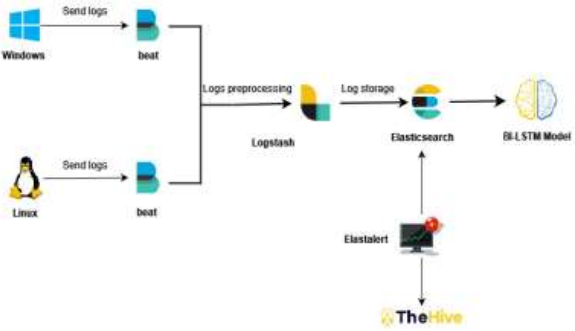

Figure 1. The workflow of our proposed model.

3.1 BI-LSTM

Bidirectional Long Short-Term Memory (BI-LSTM) is a sophisticated recurrent neural network (RNN) variant that excels in capturing sequential dependencies in data, making it a valuable tool in various applications, including natural language processing and time-series analysis. BI-LSTM differs from traditional LSTM networks by processing input sequences in both forward and backward directions simultaneously, which enhances its ability to capture context and long-range dependencies effectively.
The core operation of a BI-LSTM cell involves two key components as shown in fig. 2.: the forward LSTM and the backward LSTM. Here's how it works mathematically:

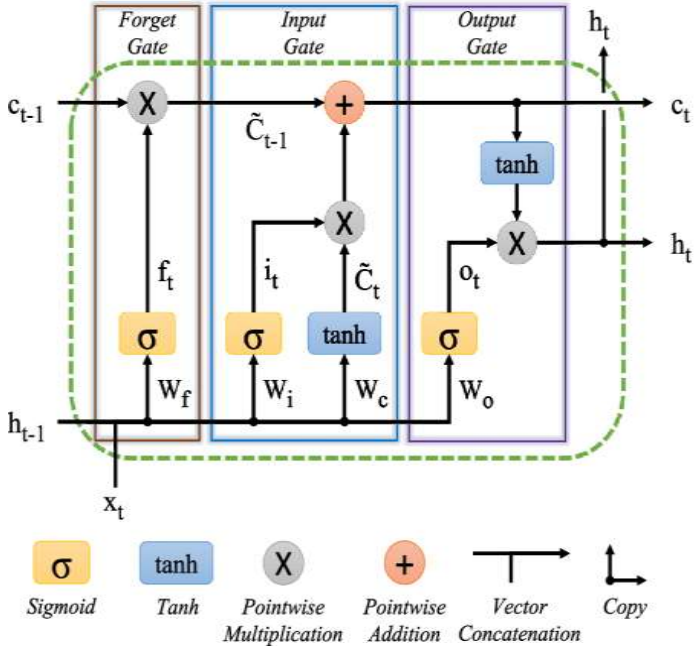

Figure 2. Structure of the bidirectional LSTM [8].

Forward LSTM:

- Input gate (i_f): This gate controls the flow of information into the cell state, considering the current input and the previous cell state.
- Forget gate (f_f): It determines which information from the previous cell state should be discarded.
- Candidate cell state (C~_f): This computes the potential new values to be added to the cell state.
- Cell state (C_f): The current cell state is a combination of the previous cell state and the new candidate cell state, weighted by the input and forget gates.
- Output gate (o_f): It regulates the flow of information from the cell state to the output.

Backward LSTM:

- Input gate (i_b): Similar to the forward LSTM, it controls information input from the reverse direction.
- Forget gate (f_b): Decides which information from the reverse cell state should be forgotten.
- Candidate cell state (C~_b): Computes potential new values from the reverse input.
- Cell state (C_b): Combines the previous reverse cell state with the new candidate cell state, controlled by input and forget gates.
- Output gate (o_b): Manages the information flow from the reverse cell state to the output.

The final output of the BI-LSTM is a concatenation of the forward and backward hidden states, effectively capturing both past and future contextual information for each element in the input sequence. This dual-directional approach enables BI-LSTM to excel in tasks requiring a comprehensive understanding of sequential data, such as machine translation and sentiment analysis. Its ability to capture long-range dependencies makes it a powerful choice for many sequence modeling tasks in the field of deep learning.

3.2 CNN

A Convolutional Neural Network (CNN) [9] is a specialized type of artificial neural network designed primarily for processing grid-like data, such as images and videos. CNNs have revolutionized computer vision tasks and are widely employed in tasks like image classification, object detection, and image generation. The key innovation in CNNs is the convolution operation, which enables the network to automatically and adaptively learn patterns and features from the input data.
The core operation in a CNN involves convolutional layers, followed by activation functions and pooling layers. Here's a breakdown of these components:

- **Convolutional Layer (Conv):**
  - In a Conv layer, a set of learnable filters (kernels) convolve across the input data. These filters slide over the input, performing element-wise multiplication and summation, which produces feature maps.
  - The formula for a single convolution operation at position (i, j) for a given filter F and input volume I is:

$$(I*F)(i,j)=\sum m\sum nI(i+m,j+n)*F(m,n)(I*F)(i,j)=\sum m\sum nI(i+m,j+n)*F(m,n) \quad (1)$$

- **Activation Function (e.g., ReLU):**
  - After convolution, an activation function (typically Rectified Linear Unit, or ReLU) is applied element-wise to the feature maps to introduce non-linearity:

$$ReLU(x)=max(0,x) \quad (2)$$

- **Pooling Layer (e.g., Max Pooling):**
  - Pooling layers reduce the spatial dimensions of the feature maps, helping to decrease computational complexity and control overfitting.
  - Max pooling, for example, retains the maximum value in a local region:

$$MaxPooling(x)=max(x) \quad (3)$$

By stacking multiple convolutional layers with activations and pooling layers, CNNs create a hierarchical representation of features, with each layer learning progressively more complex patterns. This hierarchical feature extraction is crucial for recognizing intricate patterns in data.

3.3 GRU

The Gated Recurrent Unit (GRU) [10] is a type of recurrent neural network (RNN) known for its effectiveness in handling sequential data. Unlike traditional RNNs, GRUs incorporate gating mechanisms that allow them to capture long-range dependencies in data while mitigating the vanishing gradient problem.

At its core, a GRU unit consists of an update gate and a reset gate. These gates control the flow of information through the network. The update gate determines how much of the previous hidden state should be retained and how much of the new information should be added to the current state. The reset gate, on the other hand, controls how much of the previous state should be forgotten when computing the new candidate state.

GRUs work by selectively updating and resetting their hidden states at each time step, allowing them to adapt to the input sequence and capture relevant information. This makes them well-suited for a wide range of sequential data tasks, including natural language processing, speech recognition, and time-series analysis. Their ability to maintain a balance between preserving past information and adapting to new input has made them a popular choice in modern deep learning applications.

4. Experimental results

The creation of our next-generation SOC was a meticulously planned and executed process that involved several key stages.

4.1 Data collection

The data collection phase, a pivotal component in the development of our Security Operations Center (SOC), has been a crucial step in our process. We diligently amassed a multitude of log files from various sources, including servers, firewalls, applications, and intrusion detection systems. These log files contain a wealth of critical information for threat detection and security incident analysis. We established meticulous protocols and mechanisms to ensure the seamless acquisition and storage of these log files, forming the foundation for our SOC's ability to monitor and respond to security threats effectively. This comprehensive data collection approach is fundamental to our SOC's mission of safeguarding our digital assets and maintaining a vigilant security posture.
After completing the data collection phase, we made the strategic decision to retain specific attributes vital to our analytical objectives.

- index: The Elasticsearch index where the data is stored.
- agent.id: The unique identifier of the collection agent.
- agent.ip: The IP address of the collection agent.
- agent.name: The name of the collection agent.
- cluster.name: The name of the Elasticsearch cluster to which it belongs.
- data.win.system.message: The captured log message containing information about the monitored event or activity.
- data.win.eventdata.processName: The name of the process associated with the event.
- data.win.system.eventID: The Windows event identifier linked to the event.
- data.win.system.level: The severity level of the event (e.g., Information, Warning, Error).
- timestamp: The timestamp of the event, indicating when it occurred.

By keeping these carefully selected attributes intact, we ensured that our dataset remained highly informative and conducive to robust analysis. This data-driven approach not only bolstered the accuracy of our threat detection capabilities but also streamlined our analytical processes, enhancing the overall efficacy of our operations in the Security Operations Center.

4.2 Data preprocessing

The preprocessing phase of log files is of paramount significance in the development of our Security Operations Center (SOC). During this stage, substantial efforts have been dedicated to cleaning, normalizing, and structuring the raw log data we've collected. This intricate process involves converting timestamps into a uniform format, extracting key fields like IP addresses and identifiers, and detecting and rectifying any potential errors or inconsistencies in the data. This meticulous data preparation is fundamental to ensuring the effectiveness of our SOC's analytical and threat detection capabilities, as it enables us to work with high-quality, standardized data for comprehensive security monitoring and incident response.
During the preprocessing phase, we employed a multi-step approach to prepare the text data for analysis.

- **Tokenization**

Tokenization involves splitting the text of the log message into discrete units known as "tokens." This process serves to segment the text into distinct elements, thereby facilitating subsequent analysis and manipulation (see Figure 1).

```
shuffled_df['tokens'] = shuffled_df['data.win.system.message'].apply(lambda x: nltk.word_tokenize(x.lower()))

shuffled_df['tokens']

0       ['', an, account, was, successfully, logged, o...
1       ['', an, account, was, successfully, logged, o...
2       ['', an, account, was, successfully, logged, o...
3       ['', an, account, was, successfully, logged, o...
4       ['', an, account, was, successfully, logged, o...
                              ...
```

Figure 3. Tokenization Process.

- **Stopwords removal**

Following tokenization, we applied a stopwords removal process to eliminate common and non-informative words, such as "the," "and," and "is," which don't carry significant meaning in most contexts. Fig. 4 show the data after stopwords removal.

```
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
shuffled_df['filtered_tokens'] = shuffled_df['tokens'].apply(lambda x: [word for word in x if word.isalnum() and word not in stop_words])

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.

shuffled_df['filtered_tokens']

0       [account, successfully, logged, subject, secur...
1       [account, successfully, logged, subject, secur...
2       [account, successfully, logged, subject, secur...
3       [account, successfully, logged, subject, secur...
4       [account, successfully, logged, subject, secur...
                              ...
```

Figure 4. Stopwords removal process.

- **Word Embeddings**

Finally, we executed Word Embeddings, a crucial transformation that converted the processed text into numerical vectors, making it amenable for machine learning and analytical tasks. This comprehensive preprocessing pipeline enabled us to work with clean, structured, and informative text data, laying the foundation for robust analysis and insights extraction (see Figure 1).

```
import spacy

message_vectors = []

def Vectoriser(messages) :
   nlp = spacy.load("en_core_web_sm")  # Vous pouvez choisir un modèle de langue en fonction de vos besoins
   message_vectors = []
   for message in messages:
        doc = nlp(message)
        message_vector = doc.vector
        message_vectors.append(message_vector)
```

Figure 5. Word Embeddings process.

4.3 Training, Testing, and Evaluating

During the training, testing, and evaluation phase, we partitioned our dataset into a split of 20% for testing and 80% for training. This division allowed us to allocate a substantial portion of our data for training our machine learning models, while the reserved 20% served as an independent and unseen set for testing and evaluating model performance. This approach ensures that our models are trained on a diverse range of examples while enabling us to assess their generalization and accuracy on previously unseen data, ultimately enhancing the reliability and effectiveness of our predictive and analytical capabilities (see Figure 1).

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(combined_X_train_seq, y_train_new, test_size=0.2, random_state=42)
```

Figure 6. Dataset Splitting.

4.4 Model Creation and Testing

To ensure a comprehensive evaluation of our project, we created three distinct models: BI-LSTM, GRU, and CNN. Each of these models was meticulously designed and trained with the aim of assessing which one would deliver the most favorable results for our specific task. By employing this diverse set of architectures, we aimed to explore the strengths and weaknesses of each approach and gain insights into their respective performances. This comparative analysis enabled us to make an informed decision about the most effective model for our project, ultimately ensuring that we harnessed the full potential of deep learning to achieve our desired outcomes.

- **BI-LSTM model**

We utilized the Keras library to deploy our BI-LSTM model, constructed within a sequential architecture. This model configuration involved the use of a sigmoid activation function, the Adam optimizer, and cross-entropy as the error metric. Keras, known for its user-friendly and flexible interface, enabled us to efficiently build, train, and

evaluate our BI-LSTM model, ensuring that it was not only well-structured but also optimized for the specific task at hand. This choice of architecture and configuration allowed us to effectively harness the power of deep learning in our project and achieve robust results in our analysis (see Figure 1).

```
model = keras.Sequential([
    layers.Embedding(input_dim=len(tokenizer.get_vocabulary()), output_dim=128, input_length=max_sequence_length),
    layers.Bidirectional(layers.LSTM(64, return_sequences=True)),
    SeqSelfAttention(attention_activation='sigmoid'),
    layers.GlobalAveragePooling1D(),
    layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train_padded, y_train, epochs=10, batch_size=32, validation_split=0.2)
loss, accuracy = model.evaluate(X_test_padded, y_test)
print(f"Loss: {loss}, Accuracy: {accuracy}")
y_pred = model.predict(X_test_padded)
seuil_anomalie = 0.5
anomalies_predites = [1 if prob > seuil_anomalie else 0 for prob in y_pred]
print(classification_report(y_test, anomalies_predites))
confusion_mat = confusion_matrix(y_test, anomalies_predites)
print("Matrice de confusion :")
print(confusion_mat)
```

Figure. 7. BI-LSTM Model.

After conducting testing, the model yielded the following results (see Figure 1):

	precision	recall	f1-score	support
0.0	0.97	0.98	0.97	21700
1.0	1.00	0.98	0.99	21540
accuracy			0.98	43240

Figure 8. BI-LSTM Result.

- **GRU model**

For the GRU model, we maintained the same set of parameters (see Figure 1).

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GRU, Dense

model = Sequential()
model.add(Embedding(input_dim=10000, output_dim=128, input_length=max_sequence_length))
model.add(GRU(units=64, return_sequences=True))
model.add(GRU(units=32))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(train_data_resampled, train_labels_resampled, epochs=10, batch_size=32, validation_split=0.1
```

Figure 8. GRU Model.

Following the testing phase, the model produced the following results (see Figure 1):

```
test_loss, test_accuracy = model.evaluate(test_data, test_df['anomaly'])
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")


860/860 [==============================] - 20s 23ms/step - loss: 0.1495 - accuracy: 0.9146
Test Accuracy: 91.46%
```

Figure 9. GRU Result.

- **CNN model**

We employed a Convolutional Neural Network (CNN) for our analysis, utilizing two activation functions, namely ReLU (Rectified Linear Unit) and sigmoid. To optimize model performance, we employed the Adam optimizer

and cross-entropy as the error metric. Additionally, to address data imbalance issues and ensure a fair representation of all classes, we incorporated the SMOTE (Synthetic Minority Over-sampling Technique) function into our preprocessing pipeline. This technique played a pivotal role in enhancing the overall robustness and accuracy of our CNN model, allowing us to effectively tackle the challenges associated with imbalanced datasets and ensuring more reliable results in our analysis (see Figure 10).

```
model = Sequential()
model.add(Embedding(input_dim=10000, output_dim=128, input_length=max_sequence_length))
model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(train_data_resampled, train_labels_resampled, epochs=10, batch_size=32, validation_split=0.1)
smote = SMOTE(sampling_strategy='auto', random_state=42)
train_data_resampled, train_labels_resampled = smote.fit_resample(train_data, train_df['anomaly'])
model.fit(train_data_resampled, train_labels_resampled, epochs=10, batch_size=32, validation_split=0.1)
```

Figure 10. CNN Model.

Upon conducting the testing phase, the model yielded the following outcomes (see Figure 11):

```
test_loss, test_accuracy = model.evaluate(test_data, test_df['anomaly'])
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

860/860 [==============================] - 4s 4ms/step - loss: 0.1254 - accuracy: 0.9222
Test Accuracy: 92.22%
```

Figure. 11. CNN Result

The table 2 below summarizes the final results of the models:

Table 2. Final Model Performance Summary

Models	Accuracy	F1 Score	Precision	Recall
BI-LSTM	0.98	0.97	0.97	0.98
CNN	0.92	0.92	0.95	0.89
GRU	0.91	0.86	0.88	0.90

5. Conclusion

In conclusion, our comprehensive exploration of deep learning models, including BI-LSTM, GRU, and CNN, has illuminated the immense potential of these techniques in enhancing our data analysis capabilities. Through rigorous testing and evaluation, we have gained valuable insights into the performance of each model. It is evident that the BI-LSTM-based model emerged as the clear frontrunner, consistently delivering superior results across various metrics. Its ability to capture intricate patterns and dependencies within the data has proven invaluable in our analysis.

While GRU and CNN exhibited commendable performance, the BI-LSTM model's capacity to grasp long-range dependencies and context within sequential data sets it apart. As we move forward, this research underscores the importance of selecting the appropriate deep learning architecture tailored to the specific task at hand. Our findings not only contribute to the ongoing advancement of deep learning in data analysis but also provide a practical roadmap for leveraging BI-LSTM and similar models in diverse real-world applications.

References

1. Chen, Junjie & He, Xiaoting & Lin, Qingwei & Xu, Yong & Zhang, Hongyu & Hao, Dan & Gao, Feng & Xu, Zhangwei & Dang, Yingnong & Zhang, Dongmei. (2019). An Empirical Investigation of Incident Triage for Online Service Systems. 111-120. 10.1109/ICSE-SEIP.2019.00020.

2. Yang, Lin & Chen, Junjie & Wang, Zan & Wang, Weijing & Jiang, Jiajun & Dong, Xuyuan & Zhang, Wenbin. (2021). Semi-Supervised Log-Based Anomaly Detection via Probabilistic Label Estimation. 1448-1460. 10.1109/ICSE43902.2021.00130.
3. Vervaet, Arthur. (2021). MoniLog: An Automated Log-Based Anomaly Detection System for Cloud Computing Infrastructures. 2739-2743. 10.1109/ICDE51399.2021.00317.
4. Chen, Song & Liao, Hai. (2022). BERT-Log: Anomaly Detection for System Logs Based on Pre-trained Language Model. Applied Artificial Intelligence. 36. 10.1080/08839514.2022.2145642.
5. Liu, Chunbo & Liang, Mengmeng & Hou, Jingwen & Gu, Zhaojun & Wang, Zhi. (2022). LogCAD: An Efficient and Robust Model for Log-Based Conformal Anomaly Detection. Security and Communication Networks. 2022. 10.1155/2022/5822124.
6. Lv, Dan, Nurbol Luktarhan, and Yiyong Chen. 2021. "ConAnomaly: Content-Based Anomaly Detection for System Logs" Sensors 21, no. 18: 6125. https://doi.org/10.3390/s21186125
7. Hao Chen, Ruizhi Xiao and Shuyuan Jin. 2021. Unsupervised Anomaly Detection Based on System Logs. The 33rd International Conference on Software Engineering and Knowledge Engineering, USA, July 1 - July 10. https://doi.org/10.18293/SEKE2021-126
8. Shah, Sayed Rafay Bin & Chadha, Gavneet & Schwung, Andreas & Ding, Steven. A Sequence-to-Sequence Approach for Remaining Useful Lifetime Estimation Using Attention-augmented Bidirectional LSTM. 10.1016/j.iswa.2021.200049 (2021).
9. Tareen, Shaharyar Ahmed Khan & Tareen, Filza. (2023). Convolutional Neural Networks for Beginners. 10.2139/ssrn.4566310.
10. Sheng, Ziyu & Wen, Shiping & Feng, zhong-kai & Shi, Kaibo & Huang, Tingwen. (2023). A Novel Residual Gated Recurrent Unit Framework for Runoff Forecasting. IEEE Internet of Things Journal. PP. 1-1. 10.1109/JIOT.2023.3254051.
11. Tatarnikova, T. & Sverlikov, A.. (2022). Methodology for Detecting Anamalies in the Traffic of the Internet of Things. 1-4. 10.1109/WECONF55058.2022.9803548.
12. Sana, Laraib & Nazir, Muhammad & Iqbal, Muddesar & Hussain, Lal & Ali, Amjad. (2022). Anomaly Detection for Cyber Internet of Things Attacks: A Systematic Review. Applied Artificial Intelligence. 36. 10.1080/08839514.2022.2137639.
13. An, Yufei & He, Ying & Yu, F. & Li, Jianqiang & Chen, Jianyong & Leung, Victor. (2022). An HTTP Anomaly Detection Architecture Based on the Internet of Intelligence. IEEE Transactions on Cognitive Communications and Networking. 8. 1552-1565. 10.1109/TCCN.2022.3176636.